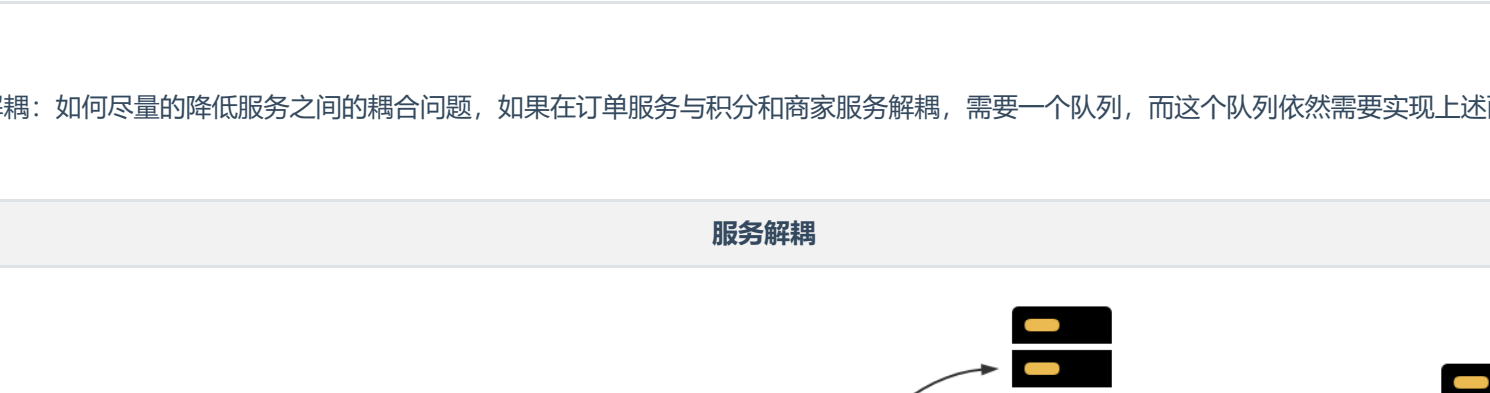
RabbitMQ

一、RabbitMQ介绍

1.1 现存问题

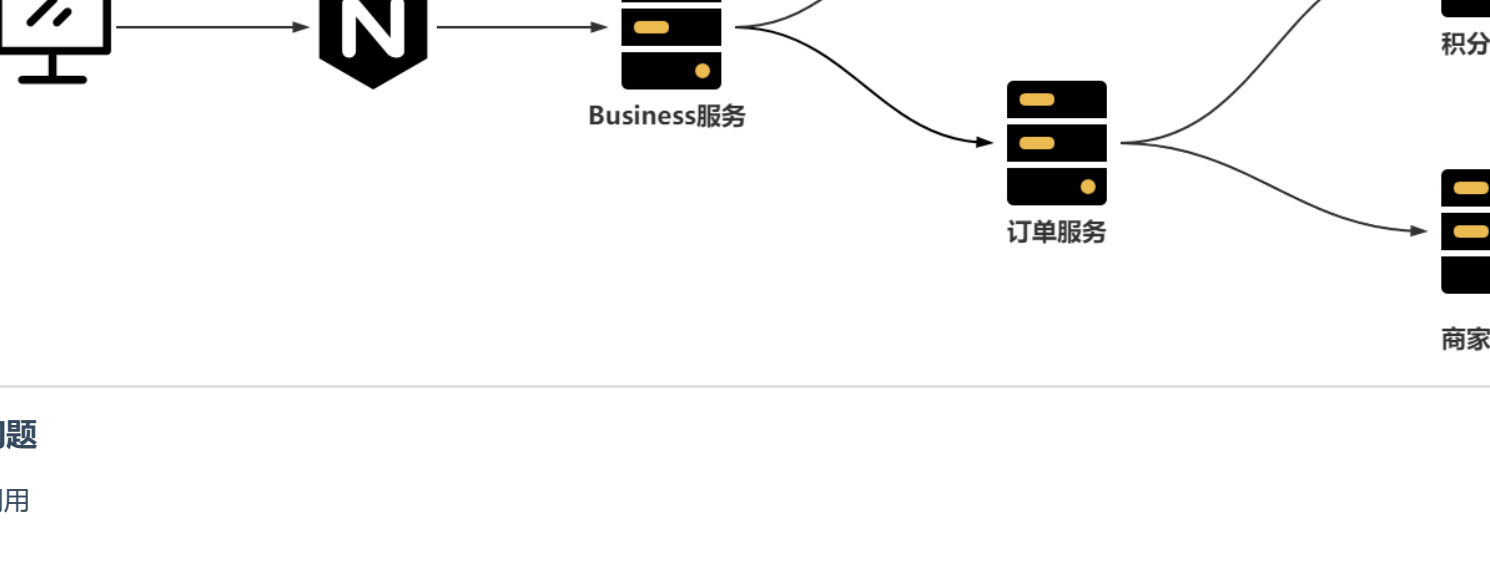
- 服务异步调用：服务A如何保证异步请求一定能被服务B接收到并处理

服务异步调用



- 削峰：海量请求，如何实现削峰的效果，将请求全部放到一个队列中，慢慢的消费，这个队列怎么实现？

削峰



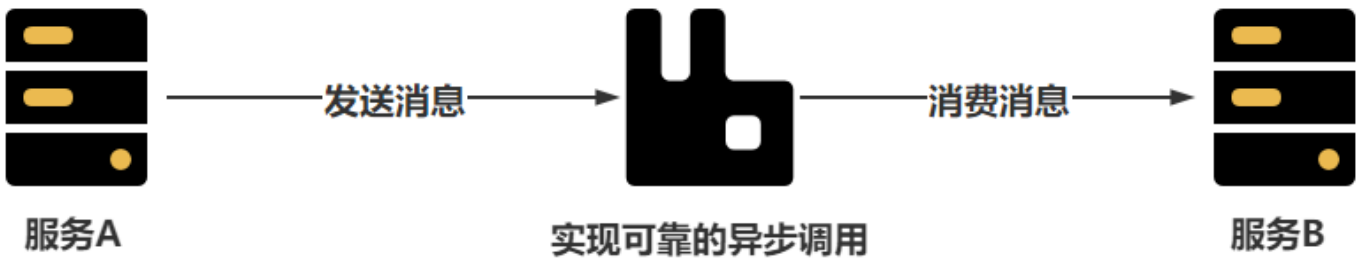
- 服务解耦：如何尽量的降低服务之间的耦合问题，如果在订单服务与积分和商家服务解耦，需要一个队列，而这个队列依然需要实现上述两种情况功能。

服务解耦

1.2 处理问题

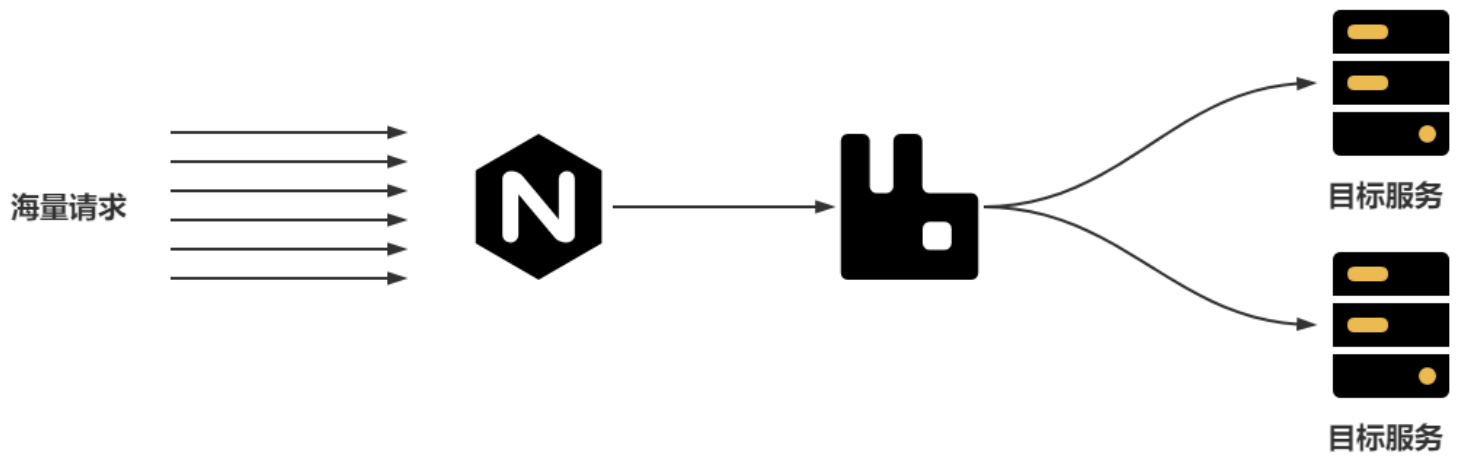
- 异步调用

异步调用



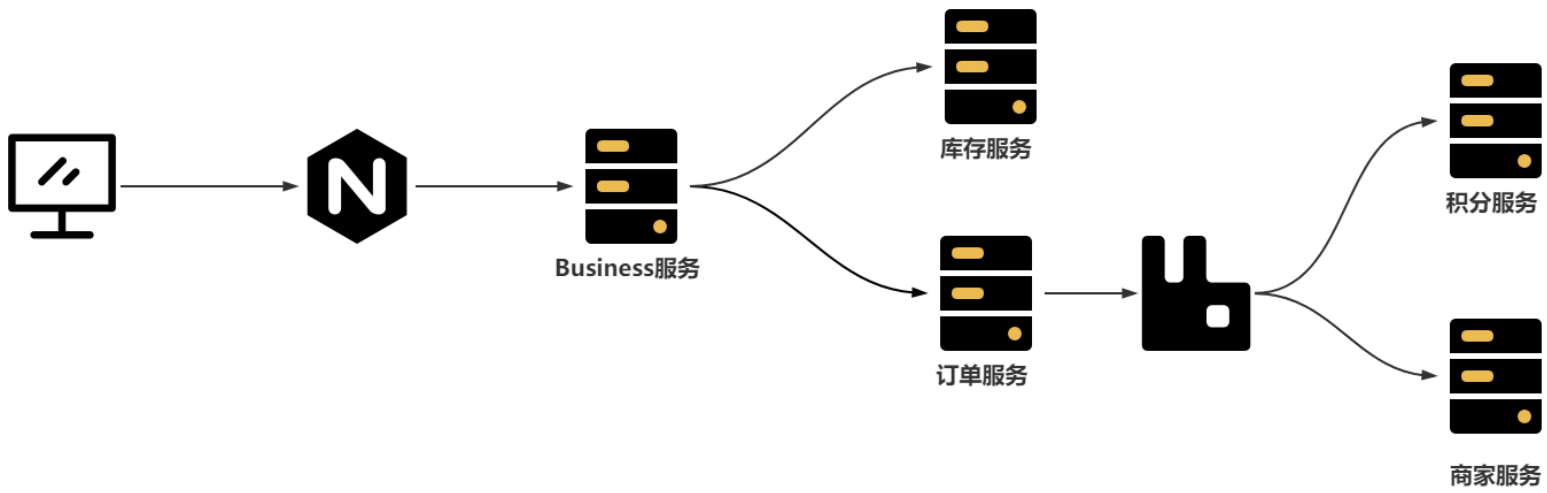
- 削峰

削峰



- 服务解耦

服务解耦



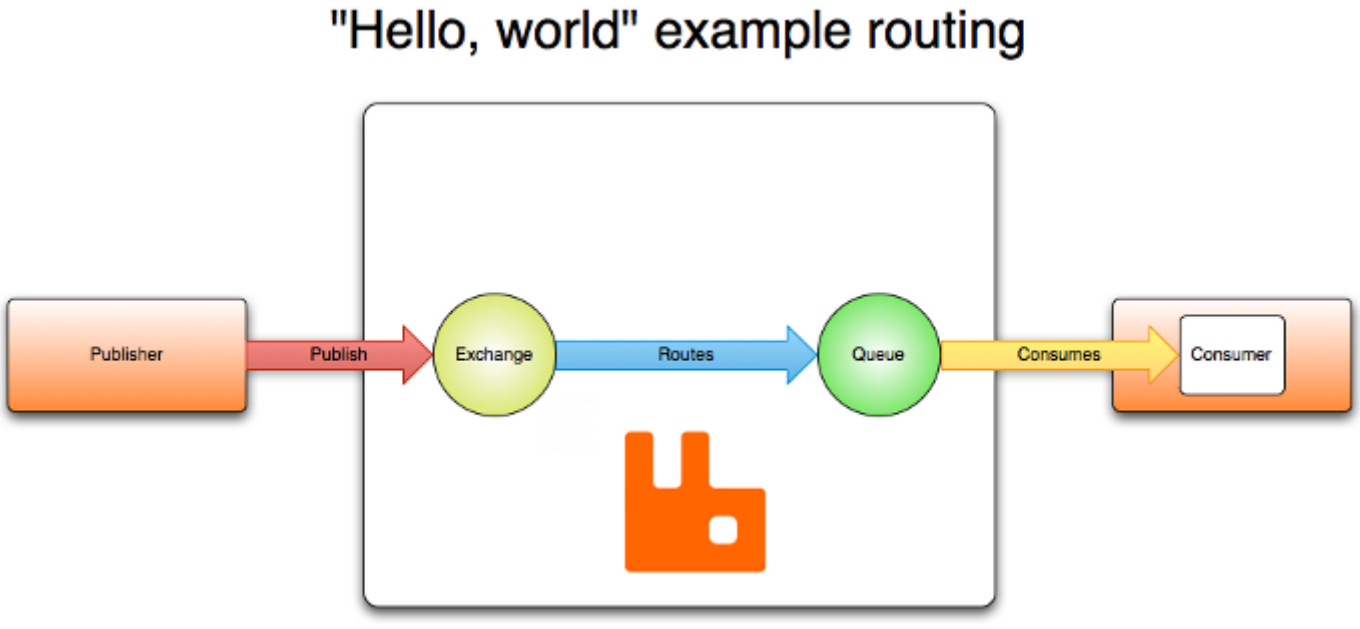
1.3 RabbitMQ介绍

百度百科：

> **RabbitMQ**是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件）。RabbitMQ服务器是用**Erlang**语言编写的，而集群和故障转移是构建在**开放电信平台**框架上的。所有主要的**编程语言**均有与代理接口通讯的**客户端**库。

AMQP协议：

AMQP



Erlang：Erlang是一种通用的面向**并发**的编程语言，Erlang充分发挥CPU的性能，延迟特别低，相比其他的MQ（Kafka，RocketMQ）延迟是最低的。

RabbitMQ支持多种语言通讯：Java，Python.....都有响应的API

RabbitMQ支持海量的插件去实现一些特殊功能，RabbitMQ自带了一款图形化界面，操作异常的简单。

二、RabbitMQ安装

2.1 Docker安装RabbitMQ

docker-compose.yml文件

```yaml
version: "3.1"
services:
  rabbitmq:
     image: daocloud.io/library/rabbitmq:3.8.5
     container_name: rabbitmq
     restart: always
     volumes:
        - ./data/:/var/lib/rabbitmq/
     ports:
        - 5672:5672
        - 15672:15672
```

在Linux内部执行：`curl localhost:5672`

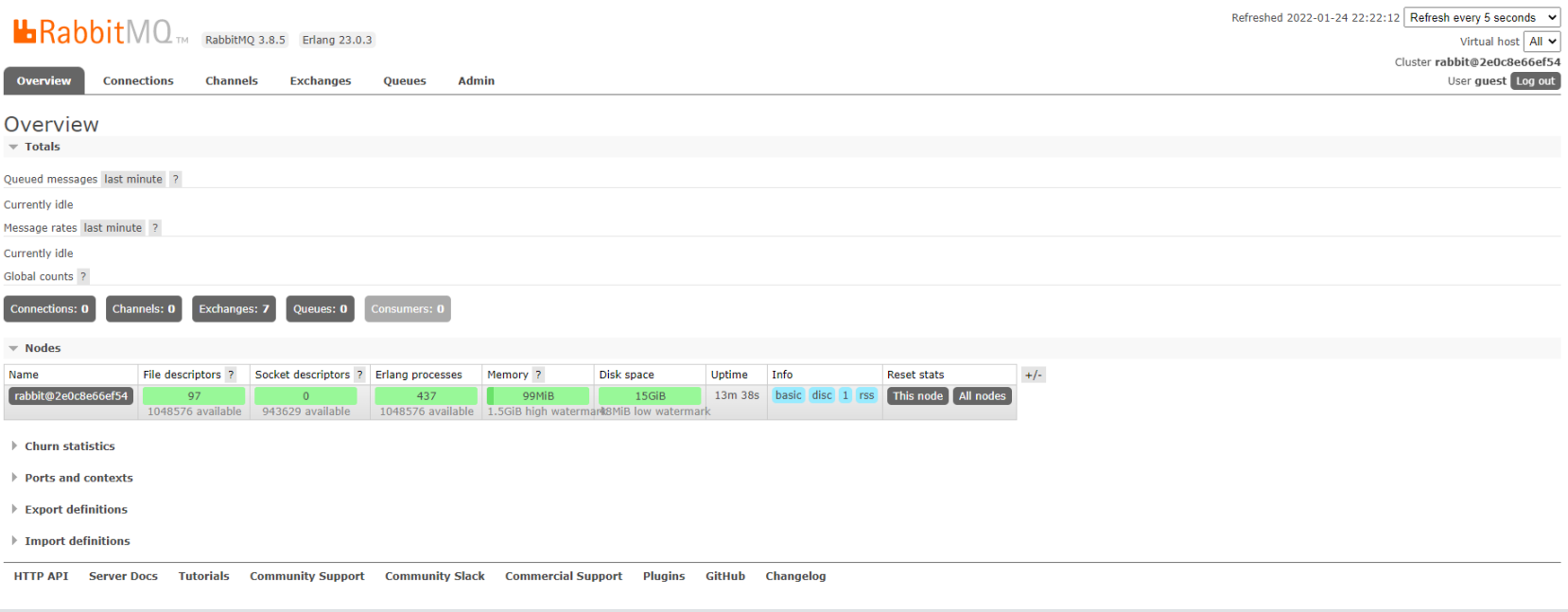
2.2 启动图形化界面

启动图形化界面

```
root@2e0c8e66ef54:/opt/rabbitmq/plugins# ls
README                              jsx-2.9.0.ez                        rabbitmq_federation_management-3.8.5.ez  rabbitmq_stomp-3.8.5.ez
accept-0.3.5.ez                     lager-3.8.0.ez                      rabbitmq_jms_topic_exchange-3.8.5.ez     rabbitmq_top-3.8.5.ez
amqp10_client-3.8.5.ez              observer_cli-1.5.4.ez               rabbitmq_management-3.8.5.ez             rabbitmq_tracing-3.8.5.ez
amqp10_common-3.8.5.ez              prometheus-4.6.0.ez                 rabbitmq_management_agent-3.8.5.ez       rabbitmq_trust_store-3.8.5.ez
amqp_client-3.8.5.ez                ra-1.1.2.ez                         rabbitmq_mqtt-3.8.5.ez                   rabbitmq_web_dispatch-3.8.5.ez
aten-0.5.3.ez                       rabbit-3.8.5.ez                     rabbitmq_peer_discovery_aws-3.8.5.ez     rabbitmq_web_mqtt-3.8.5.ez
base64url-0.0.1.ez                  rabbit_common-3.8.5.ez              rabbitmq_peer_discovery_common-3.8.5.ez  rabbitmq_web_mqtt_examples-3.8.5.ez
cowboy-2.6.1.ez                     rabbitmq_amqp1_0-3.8.5.ez           rabbitmq_peer_discovery_consul-3.8.5.ez  rabbitmq_web_stomp-3.8.5.ez
cowlib-2.7.0.ez                     rabbitmq_auth_backend_cache-3.8.5.ez rabbitmq_peer_discovery_etcd-3.8.5.ez   rabbitmq_web_stomp_examples-3.8.5.ez
credentials_obfuscation-2.0.0.ez    rabbitmq_auth_backend_http-3.8.5.ez  rabbitmq_peer_discovery_k8s-3.8.5.ez     ranch-1.7.1.ez
cuttlefish-2.2.0.ez                 rabbitmq_auth_backend_ldap-3.8.5.ez  rabbitmq_prelaunch-3.8.5.ez              recon-2.5.1.ez
eetcd-0.3.2.ez                      rabbitmq_auth_backend_oauth2-3.8.5.ez rabbitmq_prometheus-3.8.5.ez            stdout_formatter-0.2.2.ez
gen_batch_server-0.8.2.ez           rabbitmq_auth_mechanism_ssl-3.8.5.ez rabbitmq_random_exchange-3.8.5.ez        syslog-3.4.5.ez
getopt-1.0.1.ez                     rabbitmq_aws-3.8.5.ez               rabbitmq_recent_history_exchange-3.8.5.ez sysmon_handler-1.2.0.ez
goldrush-0.1.9.ez                   rabbitmq_consistent_hash_exchange-3.8.5.ez rabbitmq_sharding-3.8.5.ez
gun-1.3.2.ez                        rabbitmq_event_exchange-3.8.5.ez    rabbitmq_shovel-3.8.5.ez
jose-1.10.1.ez                      rabbitmq_federation-3.8.5.ez        rabbitmq_shovel_management-3.8.5.ez
root@2e0c8e66ef54:/opt/rabbitmq/plugins# cd ../sbin/
root@2e0c8e66ef54:/opt/rabbitmq/sbin# ls
rabbitmq-defaults  rabbitmq-diagnostics  rabbitmq-env  rabbitmq-plugins  rabbitmq-queues  rabbitmq-server  rabbitmq-upgrade  rabbitmqctl
root@2e0c8e66ef54:/opt/rabbitmq/sbin# ./rabbitmq-plugins enable rabbitmq_management
Enabling plugins on node rabbit@2e0c8e66ef54:
rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@2e0c8e66ef54...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch

started 3 plugins.
root@2e0c8e66ef54:/opt/rabbitmq/sbin#
```

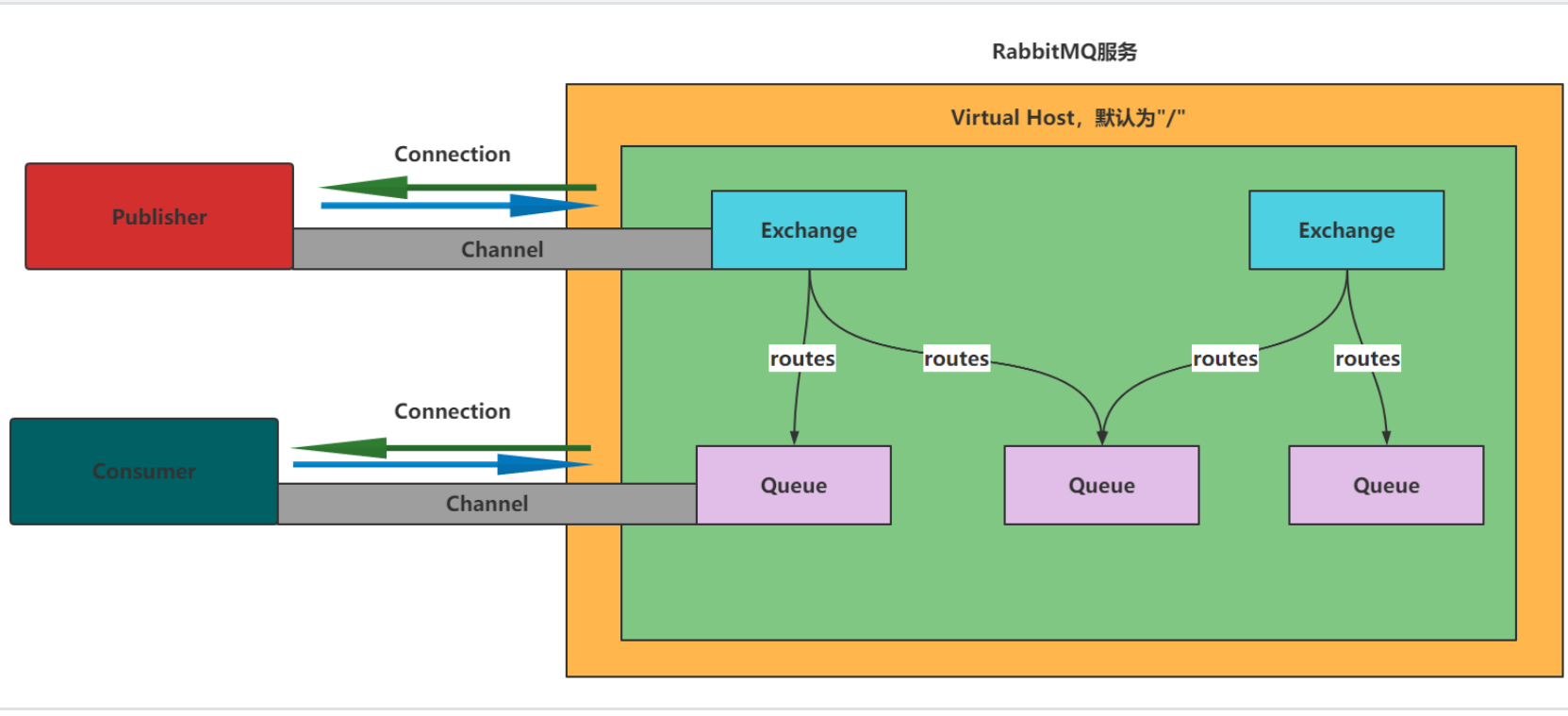
访问15672端口：默认的用户名和密码均为：`guest`

访问15672端口



三、RabbitMQ架构

完整架构



四、RabbitMQ通讯方式

4.1 RabbitMQ提供的通讯方式

- Hello World!：为了入门操作!
- Work queues：一个队列被多个消费者消费
- Publish/Subscribe：手动创建Exchange（FANOUT）
- Routing：手动创建Exchange（DIRECT）
- Topics：手动创建Exchange（TOPIC）
- RPC：RPC方式
- Publisher Confirms：保证消息可靠性

4.2 构建Connection工具类

- 导入依赖：amqp-client，junit

```
<dependencies>
      <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
            <version>5.9.0</version>
      </dependency>
      <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
      </dependency>
</dependencies>
```

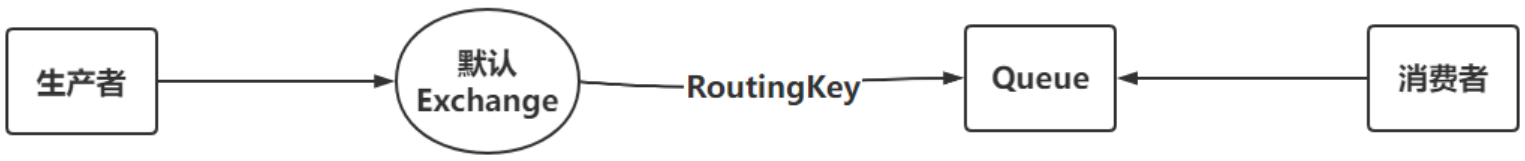
- 构建工具类：

```
package com.mashibing.util;
```

```
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

/**
 * @author zjw
 * @description
 */
public class RabbitMQConnectionUtil {

      public static final String RABBITMQ_HOST = "192.168.11.32";

      public static final int RABBITMQ_PORT = 5672;

      public static final String RABBITMQ_USERNAME = "guest";

      public static final String RABBITMQ_PASSWORD = "guest";

      public static final String RABBITMQ_VIRTUAL_HOST = "/";

      /**
       * 构建RabbitMQ的连接对象
       * @return
       */
      public static Connection getConnection() throws Exception {
            //1. 创建Connection工厂
            ConnectionFactory factory = new ConnectionFactory();

            //2. 设置RabbitMQ的连接信息
            factory.setHost(RABBITMQ_HOST);
            factory.setPort(RABBITMQ_PORT);
            factory.setUsername(RABBITMQ_USERNAME);
            factory.setPassword(RABBITMQ_PASSWORD);
            factory.setVirtualHost(RABBITMQ_VIRTUAL_HOST);

            //3. 返回连接对象
            Connection connection = factory.newConnection();
            return connection;
      }

}
```

4.3 Hello World

通讯方式



生产者:

```
package com.mashibing.helloworld;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import org.junit.Test;

/**
 * @author zjw
 * @description
 * @date 2022/1/24 22:54
 */
public class Publisher {

      public static final String QUEUE_NAME = "hello";

      @Test
      public void publish() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
```

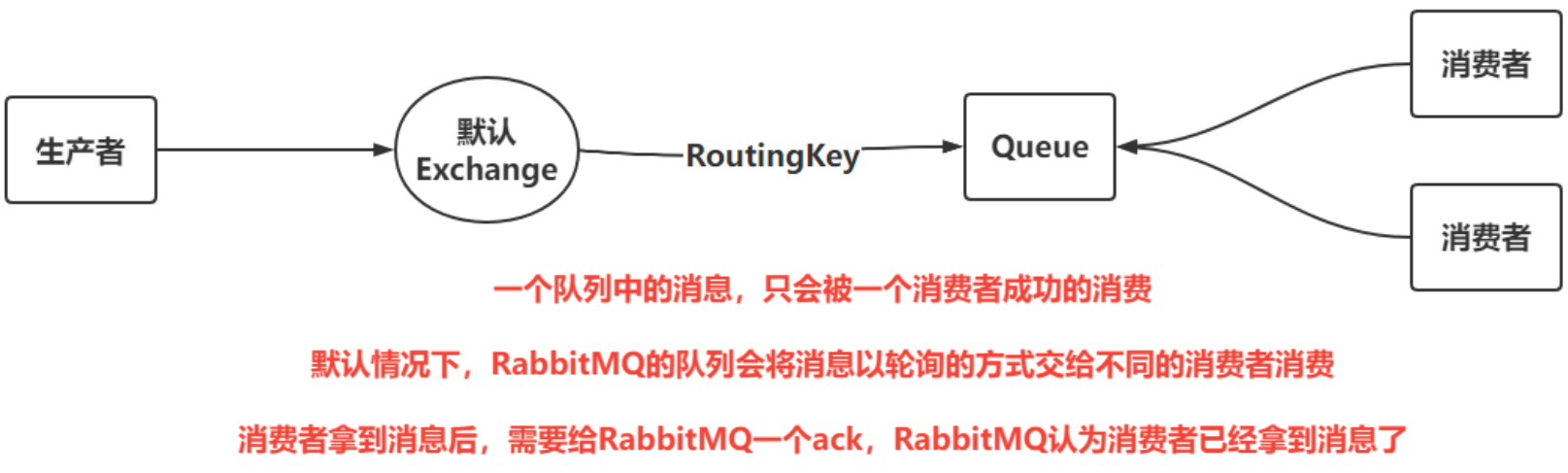
```
            Channel channel = connection.createChannel();

            //3. 构建队列
            channel.queueDeclare(QUEUE_NAME,false,false,false,null);

            //4. 发布消息
            String message = "Hello World!";
            channel.basicPublish("",QUEUE_NAME,null,message.getBytes());
            System.out.println("消息发送成功！");
    }
}
```

消费者：

```
package com.mashibing.helloworld;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.*;
import org.junit.Test;

import java.io.IOException;

/**
 * @author zjw
 * @description
 * @date 2022/1/24 23:02
 */
public class Consumer {

    @Test
    public void consume() throws Exception {
        //1. 获取连接对象
        Connection connection = RabbitMQConnectionUtil.getConnection();

        //2. 构建Channel
        Channel channel = connection.createChannel();

        //3. 构建队列
        channel.queueDeclare(Publisher.QUEUE_NAME,false,false,false,null);

        //4. 监听消息
        DefaultConsumer callback = new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者获取到消息：" + new String(body,"UTF-8"));
            }
        };
        channel.basicConsume(Publisher.QUEUE_NAME,true,callback);
        System.out.println("开始监听队列");

        System.in.read();
    }
}
```

4.4 Work Queues

WorkQueues需要学习的内容



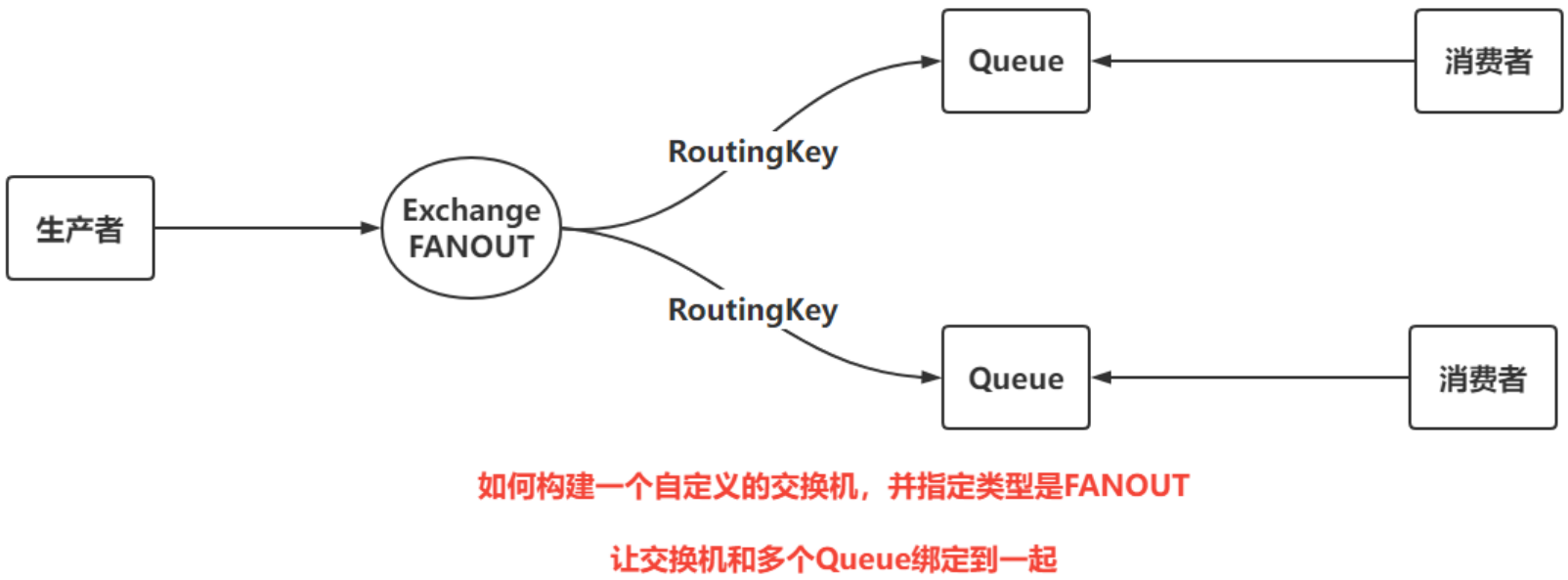
- 生产者：生产者和Hello World的形式是一样的，都是将消息推送到默认交换机。
- 消费者：让消费者关闭自动ack，并且设置消息的流控，最终实现消费者可以尽可能去多消费消息

```
package com.mashibing.workqueues;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.*;
```

```
import org.junit.Test;

import java.io.IOException;

/**
 * @author zjw
 * @description
 * @date 2022/1/25 19:52
 */
public class Consumer {

      @Test
      public void consume1() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建队列
            channel.queueDeclare(Publisher.QUEUE_NAME,false,false,false,null);

            //3.5 设置消息的流控
            channel.basicQos(3);

            //4. 监听消息
            DefaultConsumer callback = new DefaultConsumer(channel){
                  @Override
                  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException
{
                        try {
                              Thread.sleep(100);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        System.out.println("消费者1号-获取到消息：" + new String(body,"UTF-8"));
                        channel.basicAck(envelope.getDeliveryTag(),false);
                  }
            };
            channel.basicConsume(Publisher.QUEUE_NAME,false,callback);
            System.out.println("开始监听队列");

            System.in.read();
      }

      @Test
      public void consume2() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建队列
            channel.queueDeclare(Publisher.QUEUE_NAME,false,false,false,null);

            channel.basicQos(3);

            //4. 监听消息
            DefaultConsumer callback = new DefaultConsumer(channel){
                  @Override
                  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException
{
                        try {
                              Thread.sleep(1000);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        System.out.println("消费者2号-获取到消息：" + new String(body,"UTF-8"));
                        channel.basicAck(envelope.getDeliveryTag(),false);
                  }
            };
            channel.basicConsume(Publisher.QUEUE_NAME,false,callback);
            System.out.println("开始监听队列");

            System.in.read();
      }
}
```

4.5 Publish/Subscribe

自定义一个交换机

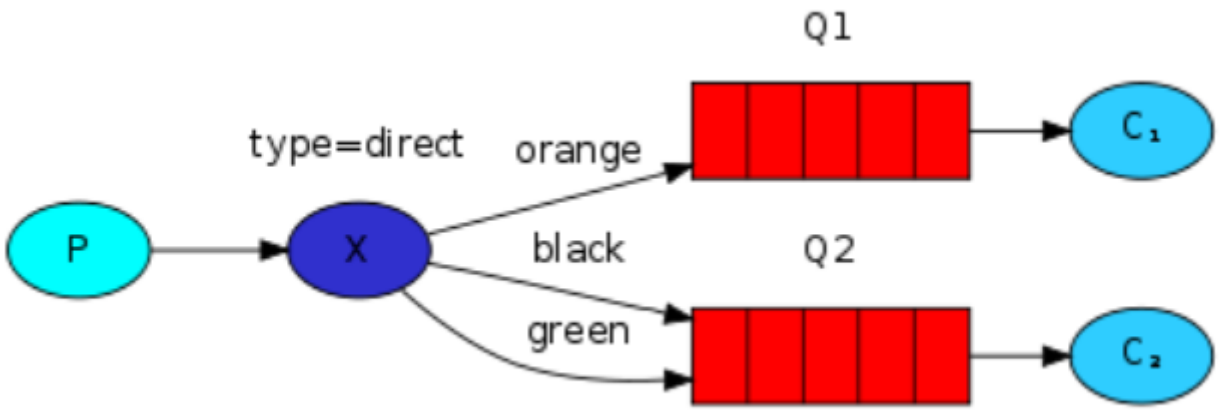


生产者：自行构建Exchange并绑定指定队列 **（FANOUT类型）**

```
package com.mashibing.pubsub;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.BuiltinExchangeType;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import org.junit.Test;

/**
 * @author zjw
 * @description
 * @date 2022/1/25 20:08
 */
public class Publisher {

      public static final String EXCHANGE_NAME = "pubsub";
      public static final String QUEUE_NAME1 = "pubsub-one";
      public static final String QUEUE_NAME2 = "pubsub-two";
      @Test
      public void publish() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建交换机
            channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.FANOUT);

            //4. 构建队列
            channel.queueDeclare(QUEUE_NAME1,false,false,false,null);
            channel.queueDeclare(QUEUE_NAME2,false,false,false,null);

            //5. 绑定交换机和队列，使用的是FANOUT类型的交换机，绑定方式是直接绑定
            channel.queueBind(QUEUE_NAME1,EXCHANGE_NAME,"");
            channel.queueBind(QUEUE_NAME2,EXCHANGE_NAME,"");

            //6. 发消息到交换机
            channel.basicPublish(EXCHANGE_NAME,"45jk6h645jk",null,"publish/subscribe!".getBytes());
            System.out.println("消息成功发送！");
      }
}
```

4.6 Routing

DIRECT类型Exchange

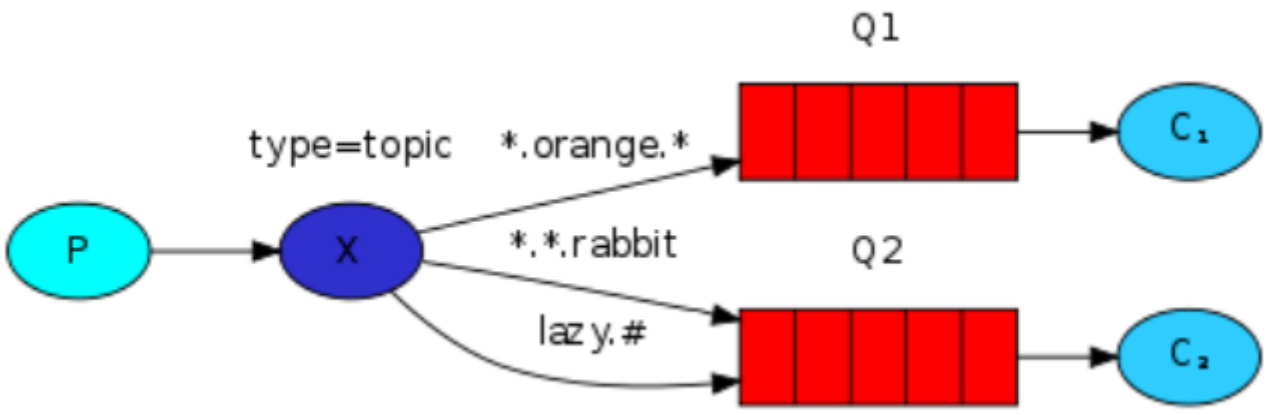


生产者：在绑定Exchange和Queue时，需要指定好routingKey，同时在发送消息时，也指定routingKey，只有routingKey一致时，才会把指定的消息路由到指定的Queue

```
package com.mashibing.routing;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.BuiltinExchangeType;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import org.junit.Test;

/**
 * @author zjw
 * @description
 * @date 2022/1/25 20:20
 */
public class Publisher {

      public static final String EXCHANGE_NAME = "routing";
      public static final String QUEUE_NAME1 = "routing-one";
      public static final String QUEUE_NAME2 = "routing-two";
      @Test
      public void publish() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建交换机
            channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT);

            //4. 构建队列
            channel.queueDeclare(QUEUE_NAME1,false,false,false,null);
            channel.queueDeclare(QUEUE_NAME2,false,false,false,null);

            //5. 绑定交换机和队列
            channel.queueBind(QUEUE_NAME1,EXCHANGE_NAME,"ORANGE");
            channel.queueBind(QUEUE_NAME2,EXCHANGE_NAME,"BLACK");
            channel.queueBind(QUEUE_NAME2,EXCHANGE_NAME,"GREEN");

            //6. 发消息到交换机
            channel.basicPublish(EXCHANGE_NAME,"ORANGE",null,"大橙子！".getBytes());
            channel.basicPublish(EXCHANGE_NAME,"BLACK",null,"黑布林大狸子".getBytes());
            channel.basicPublish(EXCHANGE_NAME,"WHITE",null,"小白兔！".getBytes());
            System.out.println("消息成功发送！");


      }

}
```

4.7 Topic

Topic模式



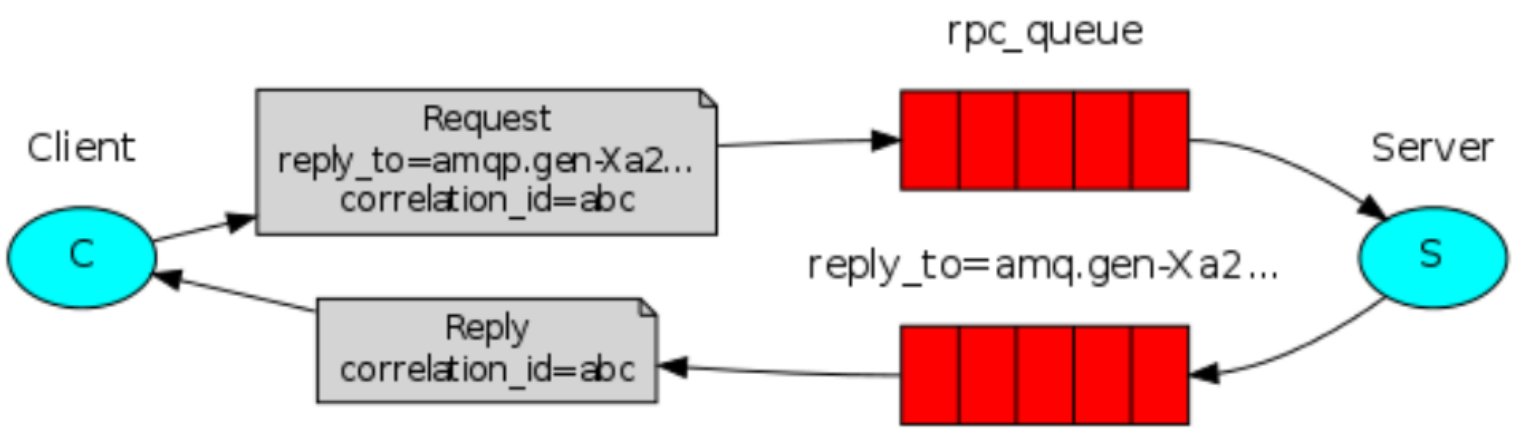
生产者：TOPIC类型可以编写带有特殊意义的routingKey的绑定方式

```
package com.mashibing.topics;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.BuiltinExchangeType;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import org.junit.Test;

/**
 * @author zjw
 * @description
 * @date 2022/1/25 20:28
 */
public class Publisher {

      public static final String EXCHANGE_NAME = "topic";
      public static final String QUEUE_NAME1 = "topic-one";
      public static final String QUEUE_NAME2 = "topic-two";
      @Test
      public void publish() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建交换机
            channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);

            //4. 构建队列
            channel.queueDeclare(QUEUE_NAME1,false,false,false,null);
            channel.queueDeclare(QUEUE_NAME2,false,false,false,null);

            //5. 绑定交换机和队列,
            // TOPIC类型的交换机在和队列绑定时，需要以aaa.bbb.ccc..方式编写routingkey
            // 其中有两个特殊字符：*（相当于占位符），#（相当通配符）
            channel.queueBind(QUEUE_NAME1,EXCHANGE_NAME,"*.orange.*");
            channel.queueBind(QUEUE_NAME2,EXCHANGE_NAME,"*.*.rabbit");
            channel.queueBind(QUEUE_NAME2,EXCHANGE_NAME,"lazy.#");

            //6. 发消息到交换机
            channel.basicPublish(EXCHANGE_NAME,"big.orange.rabbit",null,"大橙兔子！".getBytes());
            channel.basicPublish(EXCHANGE_NAME,"small.white.rabbit",null,"小白兔".getBytes());
            channel.basicPublish(EXCHANGE_NAME,"lazy.dog.dog.dog.dog.dog.dog",null,"懒狗狗狗狗狗狗".getBytes());
            System.out.println("消息成功发送！");

      }
}
```

4.8 RPC（了解）

> 因为两个服务在交互时，可以尽量做到Client和Server的解耦，通过RabbitMQ进行解耦操作
>
> 需要让Client发送消息时，携带两个属性：
>
> - replyTo告知Server将相应信息放到哪个队列
> - correlationId告知Server发送相应消息时，需要携带位置标示来告知Client响应的信息

RPC方式



客户端：

```
package com.mashibing.rpc;

import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.*;
import org.junit.Test;

import java.io.IOException;
import java.util.UUID;

/**
 * @author zjw
 * @description
 * @date 2022/2/8 20:03
 */
public class Publisher {

      public static final String QUEUE_PUBLISHER = "rpc_publisher";
      public static final String QUEUE_CONSUMER = "rpc_consumer";

      @Test
      public void publish() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建队列
            channel.queueDeclare(QUEUE_PUBLISHER,false,false,false,null);
            channel.queueDeclare(QUEUE_CONSUMER,false,false,false,null);

            //4. 发布消息
            String message = "Hello RPC!";
            String uuid = UUID.randomUUID().toString();
            AMQP.BasicProperties props = new AMQP.BasicProperties()
                        .builder()
                        .replyTo(QUEUE_CONSUMER)
                        .correlationId(uuid)
                        .build();
             channel.basicPublish("",QUEUE_PUBLISHER,props,message.getBytes());

             channel.basicConsume(QUEUE_CONSUMER,false,new DefaultConsumer(channel){
                  @Override
                  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
                        String id = properties.getCorrelationId();
                        if(id != null && id.equalsIgnoreCase(uuid)){
                              System.out.println("接收到服务端的响应：" + new String(body,"UTF-8"));
                        }
                        channel.basicAck(envelope.getDeliveryTag(),false);
                  }
            });
             System.out.println("消息发送成功！");

             System.in.read();
      }


}
```

服务端：

```
package com.mashibing.rpc;
```

```
import com.mashibing.helloworld.Publisher;
import com.mashibing.util.RabbitMQConnectionUtil;
import com.rabbitmq.client.*;
import org.junit.Test;

import java.io.IOException;

/**
 * @author zjw
 * @description
 * @date 2022/1/24 23:02
 */
public class Consumer {

      public static final String QUEUE_PUBLISHER = "rpc_publisher";
      public static final String QUEUE_CONSUMER = "rpc_consumer";

      @Test
      public void consume() throws Exception {
            //1. 获取连接对象
            Connection connection = RabbitMQConnectionUtil.getConnection();

            //2. 构建Channel
            Channel channel = connection.createChannel();

            //3. 构建队列
            channel.queueDeclare(QUEUE_PUBLISHER,false,false,false,null);
            channel.queueDeclare(QUEUE_CONSUMER,false,false,false,null);


            //4. 监听消息
            DefaultConsumer callback = new DefaultConsumer(channel){
                  @Override
                  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
                        System.out.println("消费者获取到消息: " + new String(body,"UTF-8"));
                        String resp = "获取到了client发出的请求，这里是响应的信息";
                        String respQueueName = properties.getReplyTo();
                        String uuid = properties.getCorrelationId();
                        AMQP.BasicProperties props = new AMQP.BasicProperties()
                                    .builder()
                                    .correlationId(uuid)
                                    .build();
                        channel.basicPublish("",respQueueName,props,resp.getBytes());
                        channel.basicAck(envelope.getDeliveryTag(),false);
                  }
            };
            channel.basicConsume(QUEUE_PUBLISHER,false,callback);
            System.out.println("开始监听队列");

            System.in.read();
      }
}
```

五、SpringBoot操作RabbitMQ

5.1 SpringBoot声明信息

- 创建项目
- 导入依赖

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

- 配置RabbitMQ信息

```
spring:
  rabbitmq:
     host: 192.168.11.32
     port: 5672
     username: guest
     password: guest
     virtual-host: /
```

- 声明交换机&队列

```
package com.mashibing.rabbitmqboot.config;
```

```
import org.springframework.amqp.core.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author zjw
 * @description
 * @date 2022/2/8 20:25
 */
@Configuration
public class RabbitMQConfig {

      public static final String EXCHANGE = "boot-exchange";
      public static final String QUEUE = "boot-queue";
      public static final String ROUTING_KEY = "*.black.*";


      @Bean
      public Exchange bootExchange(){
            // channel.DeclareExchange
            return ExchangeBuilder.topicExchange(EXCHANGE).build();
      }

      @Bean
      public Queue bootQueue(){
            return QueueBuilder.durable(QUEUE).build();
      }

      @Bean
      public Binding bootBinding(Exchange bootExchange,Queue bootQueue){
            return BindingBuilder.bind(bootQueue).to(bootExchange).with(ROUTING_KEY).noargs();
      }
}
```

5.2 生产者操作

```
package com.mashibing.rabbitmqboot;

import com.mashibing.rabbitmqboot.config.RabbitMQConfig;
import org.junit.jupiter.api.Test;
import org.springframework.amqp.AmqpException;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessagePostProcessor;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

/**
 * @author zjw
 * @description
 * @date 2022/2/8 21:05
 */
@SpringBootTest
public class PublisherTest {

      @Autowired
      public RabbitTemplate rabbitTemplate;

      @Test
      public void publish(){
            rabbitTemplate.convertAndSend(RabbitMQConfig.EXCHANGE,"big.black.dog","message");
            System.out.println("消息发送成功");
      }


      @Test
      public void publishWithProps(){
            rabbitTemplate.convertAndSend(RabbitMQConfig.EXCHANGE, "big.black.dog", "messageWithProps", new MessagePostProcessor() {
                  @Override
                  public Message postProcessMessage(Message message) throws AmqpException {
                        message.getMessageProperties().setCorrelationId("123");
                        return message;
                  }
            });
            System.out.println("消息发送成功");
      }
}
```

5.3 消费者操作

```java
package com.mashibing.rabbitmqboot;

import com.mashibing.rabbitmqboot.config.RabbitMQConfig;
import com.rabbitmq.client.Channel;
import org.junit.jupiter.api.Test;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Component;

import java.io.IOException;

/**
 * @author zjw
 * @description
 * @date 2022/2/8 21:11
 */
@Component
public class ConsumeListener {

      @RabbitListener(queues = RabbitMQConfig.QUEUE)
      public void consume(String msg, Channel channel, Message message) throws IOException {
            System.out.println("队列的消息为：" + msg);
            String correlationId = message.getMessageProperties().getCorrelationId();
            System.out.println("唯一标识为：" + correlationId);
            channel.basicAck(message.getMessageProperties().getDeliveryTag(),false);
      }
}
```

六、RabbitMQ保证消息可靠性

6.1 保证消息一定送达到Exchange

Confirm机制

可以通过Confirm效果保证消息一定送达到Exchange，官方提供了三种方式，选择了对于效率影响最低的异步回调的效果

```java
//4. 开启confirms
channel.confirmSelect();

//5. 设置confirms的异步回调
channel.addConfirmListener(new ConfirmListener() {
      @Override
      public void handleAck(long deliveryTag, boolean multiple) throws IOException {
            System.out.println("消息成功的发送到Exchange！");
      }

      @Override
      public void handleNack(long deliveryTag, boolean multiple) throws IOException {
            System.out.println("消息没有发送到Exchange，尝试重试，或者保存到数据库做其他补偿操作！");
      }
});
```

6.2 保证消息可以路由到Queue

Return机制

为了保证Exchange上的消息一定可以送达到Queue

```java
//6. 设置Return回调，确认消息是否路由到了Queue
channel.addReturnListener(new ReturnListener() {
      @Override
      public void handleReturn(int replyCode, String replyText, String exchange, String routingKey, AMQP.BasicProperties properties, byte[] body) throws
IOException {
            System.out.println("消息没有路由到指定队列，做其他的补偿措施！！");
      }
});
//7. 在发送消息时，将basicPublish方法参数中的mandatory设置为true，即可开启Return机制，当消息没有路由到队列中时，就会执行return回调
```

6.3 保证Queue可以持久化消息

DeliveryMode设置消息持久化

DeliveryMode设置为2代表持久化，如果设置为1，就代表不会持久化。

```
//7. 设置消息持久化
AMQP.BasicProperties props = new AMQP.BasicProperties()
      .builder()
      .deliveryMode(2)
      .build();

//7. 发布消息
channel.basicPublish("","confirms",true,props,message.getBytes());
```

6.4 保证消费者可以正常消费消息

详情看WorkQueue模式

6.5 SpringBoot实现上述操作

6.5.1 Confirm

- 编写配置文件开启Confirm机制

```
spring:
  rabbitmq:
     publisher-confirm-type: correlated   # 新版本
     publisher-confirms: true   # 老版本
```

- 在发送消息时，配置RabbitTemplate

```
@Test
public void publishWithConfirms() throws IOException {
      rabbitTemplate.setConfirmCallback(new RabbitTemplate.ConfirmCallback() {
            @Override
            public void confirm(CorrelationData correlationData, boolean ack, String cause) {
                  if(ack){
                        System.out.println("消息已经送达到交换机！！");
                  }else{
                        System.out.println("消息没有送达到Exchange，需要做一些补偿操作！！retry！！！");
                  }
            }
      });
      rabbitTemplate.convertAndSend(RabbitMQConfig.EXCHANGE,"big.black.dog","message");
      System.out.println("消息发送成功");

      System.in.read();
}
```

6.5.2 Return

- 编写配置文件开启Return机制

```
spring:
  rabbitmq:
     publisher-returns: true # 开启Return机制
```

- 在发送消息时，配置RabbitTemplate

```
@Test
public void publishWithReturn() throws IOException {
      // 新版本用 setReturnsCallback ，老版本用setReturnCallback
      rabbitTemplate.setReturnsCallback(new RabbitTemplate.ReturnsCallback() {
            @Override
            public void returnedMessage(ReturnedMessage returned) {
                  String msg = new String(returned.getMessage().getBody());
                  System.out.println("消息：" + msg + "路由队列失败！！做补救操作！！");
            }
      });
      rabbitTemplate.convertAndSend(RabbitMQConfig.EXCHANGE,"big.black.dog","message");
      System.out.println("消息发送成功");

      System.in.read();
}
```

6.5.3 消息持久化

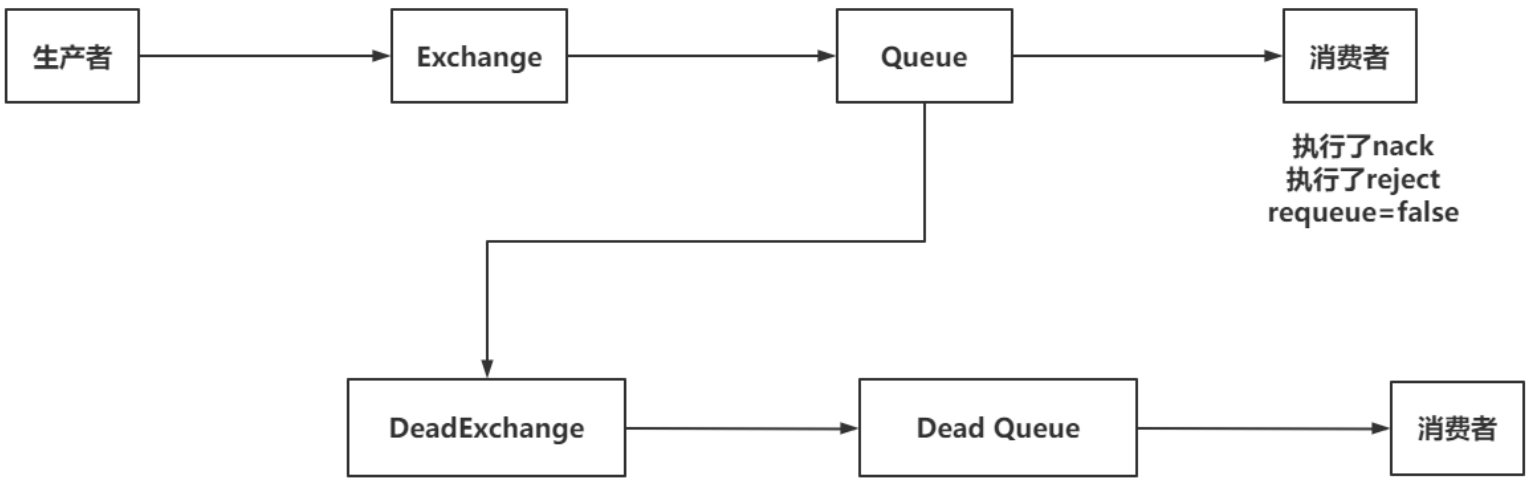
```
@Test
public void publishWithBasicProperties() throws IOException {
      rabbitTemplate.convertAndSend(RabbitMQConfig.EXCHANGE, "big.black.dog", "message", new MessagePostProcessor() {
            @Override
            public Message postProcessMessage(Message message) throws AmqpException {
                  // 设置消息的持久化!
                  message.getMessageProperties().setDeliveryMode(MessageDeliveryMode.PERSISTENT);
                  return message;
            }
      });
       System.out.println("消息发送成功");
}
```

七、RabbitMQ死信队列&延迟交换机

7.1 什么是死信

死信&死信队列

1：消息被消费者拒绝，requeue设置为false
2.1：发送消息时设置消息的生存时间，如果生存时间到了，还没有被消费。
2.2：也可以指定某个队列中所有消息的生存时间，如果生存时间到了，还没有被消费。
3：队列已经达到消息的最大长度后，再路由过来的消息直接变为死信



死信队列的应用：

- 基于死信队列在队列消息已满的情况下，消息也不会丢失
- 实现延迟消费的效果。比如：下订单时，有15分钟的付款时间

7.2 实现死信队列

7.2.1 准备Exchange&Queue

```
package com.mashibing.rabbitmqboot.config;

import org.springframework.amqp.core.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author zjw
 * @description
 * @date 2022/2/10 15:04
 */
@Configuration
public class DeadLetterConfig {

      public static final String NORMAL_EXCHANGE = "normal-exchange";
      public static final String NORMAL_QUEUE = "normal-queue";
      public static final String NORMAL_ROUTING_KEY = "normal.#";

      public static final String DEAD_EXCHANGE = "dead-exchange";
      public static final String DEAD_QUEUE = "dead-queue";
      public static final String DEAD_ROUTING_KEY = "dead.#";


      @Bean
      public Exchange normalExchange(){
            return ExchangeBuilder.topicExchange(NORMAL_EXCHANGE).build();
      }
```

```
    @Bean
    public Queue normalQueue(){
        return QueueBuilder.durable(NORMAL_QUEUE).deadLetterExchange(DEAD_EXCHANGE).deadLetterRoutingKey("dead.abc").build();
    }

    @Bean
    public Binding normalBinding(Queue normalQueue,Exchange normalExchange){
        return BindingBuilder.bind(normalQueue).to(normalExchange).with(NORMAL_ROUTING_KEY).noargs();
    }


    @Bean
    public Exchange deadExchange(){
        return ExchangeBuilder.topicExchange(DEAD_EXCHANGE).build();
    }

    @Bean
    public Queue deadQueue(){
        return QueueBuilder.durable(DEAD_QUEUE).build();
    }

    @Bean
    public Binding deadBinding(Queue deadQueue,Exchange deadExchange){
        return BindingBuilder.bind(deadQueue).to(deadExchange).with(DEAD_ROUTING_KEY).noargs();
    }

}
```

7.2.2 实现效果

- 基于消费者进行reject或者nack实现死信效果

```
package com.mashibing.rabbitmqboot;

import com.mashibing.rabbitmqboot.config.DeadLetterConfig;
import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

import java.io.IOException;

/**
 * @author zjw
 * @description
 * @date 2022/2/10 15:17
 */
@Component
public class DeadListener {

    @RabbitListener(queues = DeadLetterConfig.NORMAL_QUEUE)
    public void consume(String msg, Channel channel, Message message) throws IOException {
        System.out.println("接收到normal队列的消息：" + msg);
        channel.basicReject(message.getMessageProperties().getDeliveryTag(),false);
        channel.basicNack(message.getMessageProperties().getDeliveryTag(),false,false);
    }
}
```

- 消息的生存时间
  - 给消息设置生存时间

```
@Test
public void publishExpire(){
    String msg = "dead letter expire";
    rabbitTemplate.convertAndSend(DeadLetterConfig.NORMAL_EXCHANGE, "normal.abc", msg, new MessagePostProcessor() {
        @Override
        public Message postProcessMessage(Message message) throws AmqpException {
            message.getMessageProperties().setExpiration("5000");
            return message;
        }
    });
}
```

  - 给队列设置消息的生存时间

```
@Bean
public Queue normalQueue(){
    return QueueBuilder.durable(NORMAL_QUEUE)
            .deadLetterExchange(DEAD_EXCHANGE)
            .deadLetterRoutingKey("dead.abc")
            .ttl(10000)
            .build();
}
```

- 设置Queue中的消息最大长度

```
@Bean
public Queue normalQueue(){
      return QueueBuilder.durable(NORMAL_QUEUE)
                  .deadLetterExchange(DEAD_EXCHANGE)
                  .deadLetterRoutingKey("dead.abc")
                  .maxLength(1)
                  .build();
}
```

只要Queue中已经有一个消息，如果再次发送一个消息，这个消息会变为死信！

7.3 延迟交换机

下载地址： https://github.com/rabbitmq/rabbitmq-delayed-message-exchange/releases/tag/3.8.9

死信队列实现延迟消费时，如果延迟时间比较复杂，比较多，直接使用死信队列时，需要创建大量的队列还对应不同的时间，可以采用延迟交换机来解决这个问题。

- 构建延迟交换机

```
package com.mashibing.rabbitmqboot.config;

import org.springframework.amqp.core.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;

/**
 * @author zjw
 * @description
 */
@Configuration
public class DelayedConfig {

      public static final String DELAYED_EXCHANGE = "delayed-exchange";
      public static final String DELAYED_QUEUE = "delayed-queue";
      public static final String DELAYED_ROUTING_KEY = "delayed.#";

      @Bean
      public Exchange delayedExchange(){
            Map<String, Object> arguments = new HashMap<>();
            arguments.put("x-delayed-type","topic");
            Exchange exchange = new CustomExchange(DELAYED_EXCHANGE,"x-delayed-message",true,false,arguments);
            return exchange;
      }

      @Bean
      public Queue delayedQueue(){
            return QueueBuilder.durable(DELAYED_QUEUE).build();
      }

      @Bean
      public Binding delayedBinding(Queue delayedQueue,Exchange delayedExchange){
            return BindingBuilder.bind(delayedQueue).to(delayedExchange).with(DELAYED_ROUTING_KEY).noargs();
      }
}
```

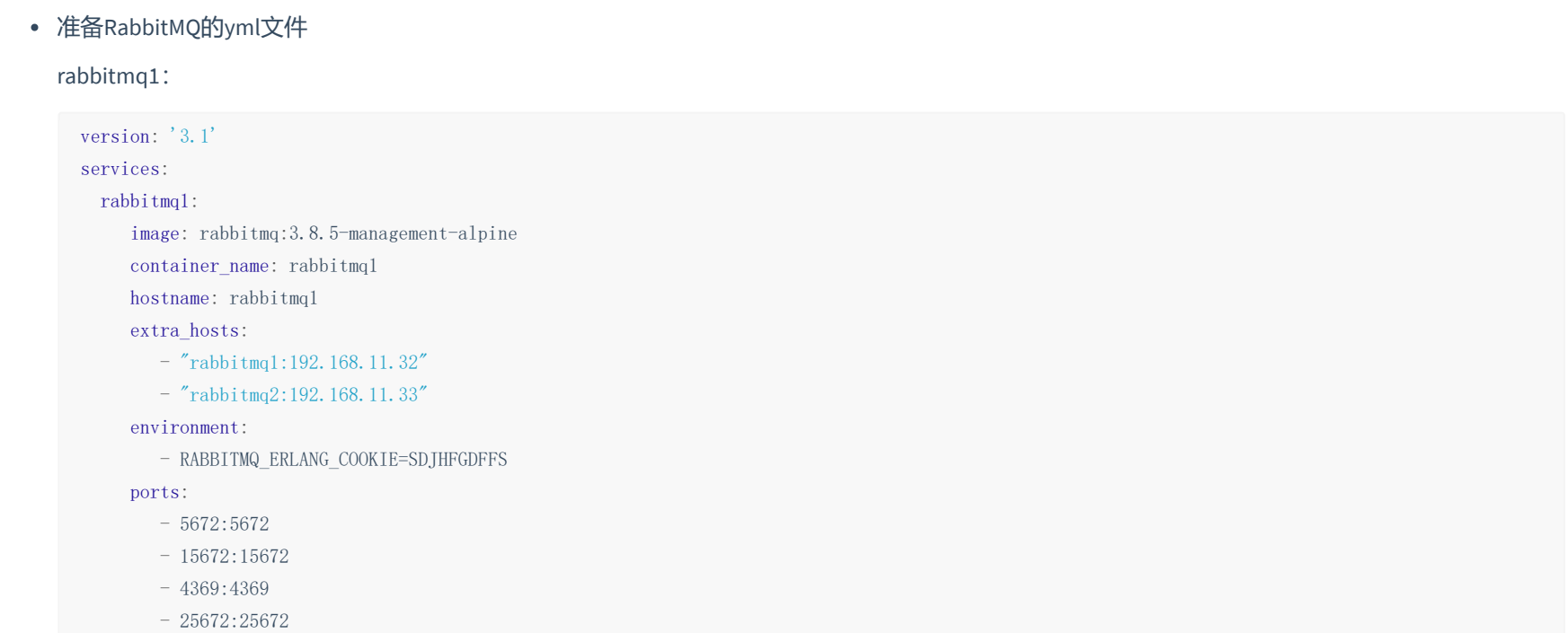
- 发送消息

```
package com.mashibing.rabbitmqboot;

import com.mashibing.rabbitmqboot.config.DelayedConfig;
import org.junit.jupiter.api.Test;
import org.springframework.amqp.AmqpException;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessagePostProcessor;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

/**
 * @author zjw
 * @description
 */
@SpringBootTest
public class DelayedPublisherTest {

      @Autowired
      private RabbitTemplate rabbitTemplate;
```

```
    @Test
    public void publish(){
        rabbitTemplate.convertAndSend(DelayedConfig.DELAYED_EXCHANGE, "delayed.abc", "xxxx", new MessagePostProcessor() {
            @Override
            public Message postProcessMessage(Message message) throws AmqpException {
                message.getMessageProperties().setDelay(30000);
                return message;
            }
        });
    }
}
```

八、RabbitMQ的集群

RabbitMQ的镜像模式



高可用

提升RabbitMQ的效率

搭建RabbitMQ集群

- 准备两台虚拟机（克隆）
- 准备RabbitMQ的yml文件

  rabbitmq1：

```
version: '3.1'
services:
  rabbitmq1:
    image: rabbitmq:3.8.5-management-alpine
    container_name: rabbitmq1
    hostname: rabbitmq1
    extra_hosts:
      - "rabbitmq1:192.168.11.32"
      - "rabbitmq2:192.168.11.33"
    environment:
      - RABBITMQ_ERLANG_COOKIE=SDJHFGDFFS
    ports:
      - 5672:5672
      - 15672:15672
      - 4369:4369
      - 25672:25672
```

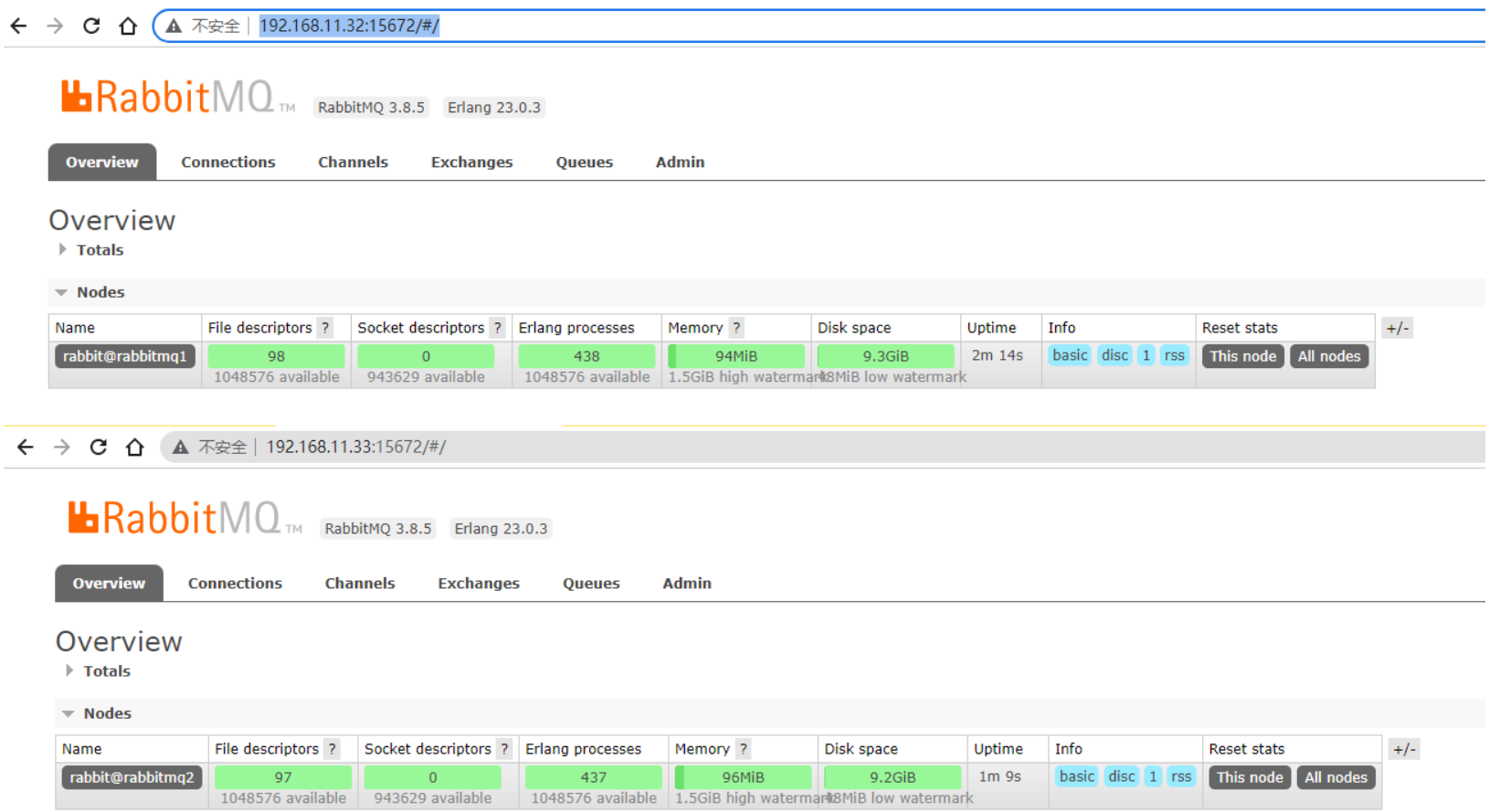
rabbitmq2：

```
version: '3.1'
services:
  rabbitmq2:
    image: rabbitmq:3.8.5-management-alpine
    container_name: rabbitmq2
    hostname: rabbitmq2
    extra_hosts:
      - "rabbitmq1:192.168.11.32"
      - "rabbitmq2:192.168.11.33"
    environment:
      - RABBITMQ_ERLANG_COOKIE=SDJHFGDFFS
    ports:
```

```
- 5672:5672
- 15672:15672
- 4369:4369
- 25672:25672
```

准备完毕之后，启动两台RabbitMQ

启动效果



- 让RabbitMQ服务实现join操作

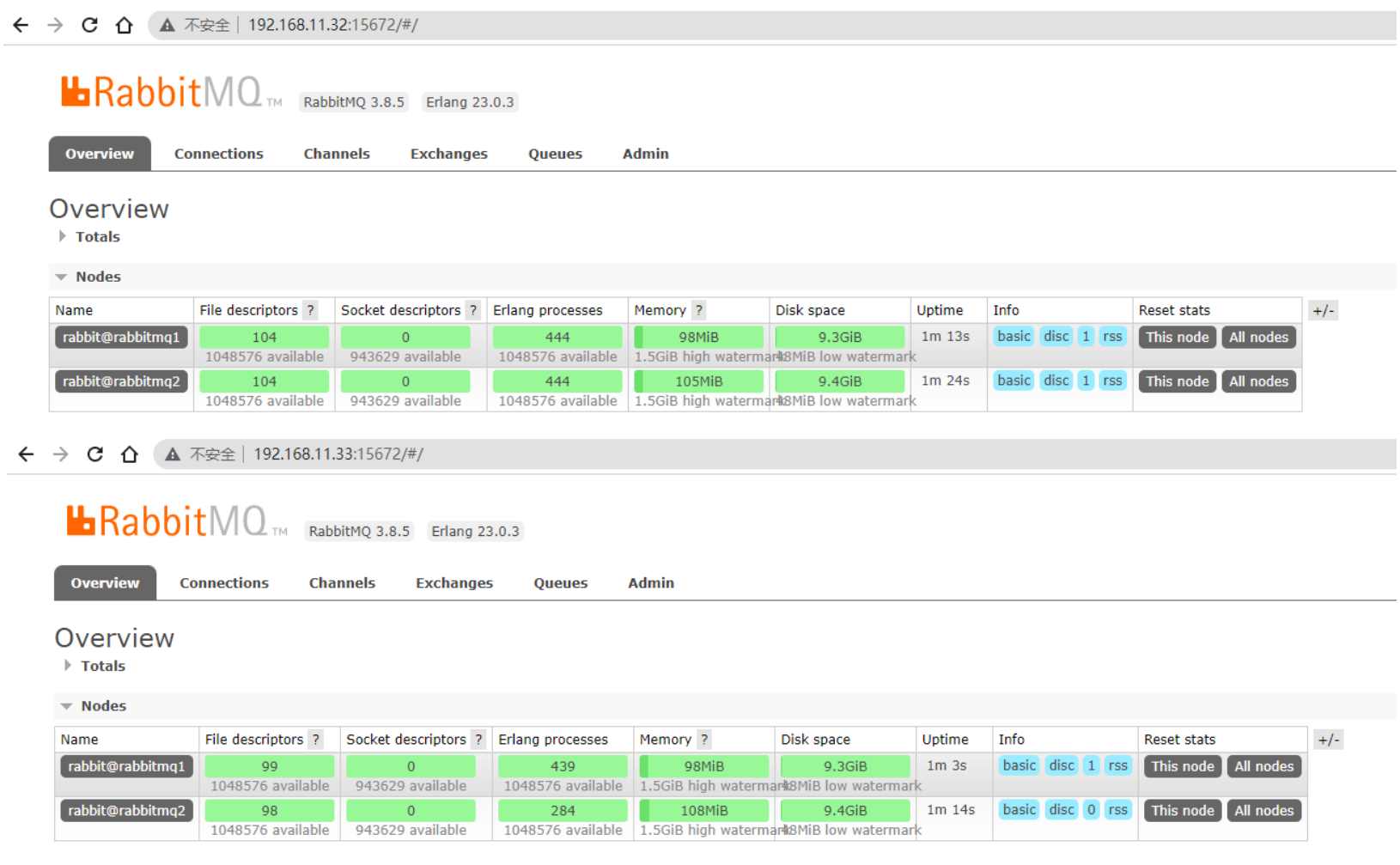
  需要四个命令完成join操作

  让rabbitmq2 join rabbitmq1，需要进入到rabbitmq2的容器内部，去执行下述命令

```
rabbitmqctl stop_app
rabbitmqctl reset
rabbitmqctl join_cluster rabbit@rabbitmq1
rabbitmqctl start_app
```

  执行成功后：

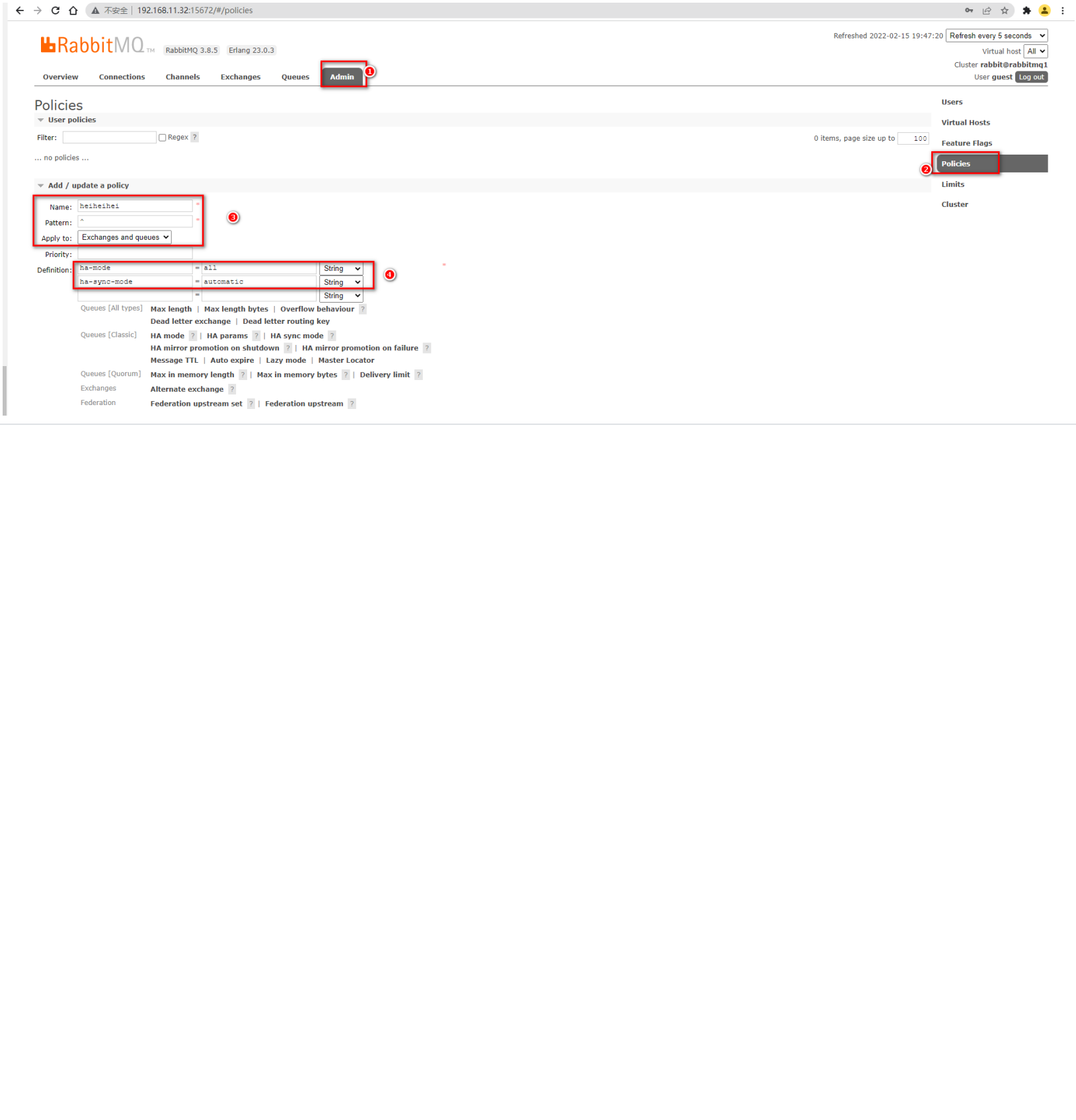
执行成功后



- 设置镜像模式

  在指定的RabbitMQ服务中设置好镜像策略即可

镜像模式


不安全 | 192.168.11.32:15672/#/policies
RabbitMQ
RabbitMQ 3.8.5
Erlang 23.0.3
Refreshed 2022-02-15 19:47:20
Refresh every 5 seconds
Virtual host All
Cluster rabbit@rabbitmq1
User guest
Log out
Overview
Connections
Channels
Exchanges
Queues
Admin
1
Policies
User policies
Filter:
Regex
0 items, page size up to 100
... no policies ...
Add / update a policy
Name: heiheihei
Pattern: ^
Apply to: Exchanges and queues
Priority:
Definition:
ha-mode = all String
ha-sync-mode = automatic String
String
3
4
Queues [All types] Max length | Max length bytes | Overflow behaviour
Dead letter exchange | Dead letter routing key
Queues [Classic] HA mode | HA params | HA sync mode
HA mirror promotion on shutdown | HA mirror promotion on failure
Message TTL | Auto expire | Lazy mode | Master Locator
Queues [Quorum] Max in memory length | Max in memory bytes | Delivery limit
Exchanges Alternate exchange
Federation Federation upstream set | Federation upstream
Users
Virtual Hosts
Feature Flags
Policies
2
Limits
Cluster