

一、消息队列介绍

1.1 同步调用与异步调用

同步调用

Feign客户端可以实现服务间的通信，但是Feign是同步调用，也就是说A服务调用B服务之后，会进入阻塞/等待状态，直到B服务返回调用结果给A服务，A服务才会继续往后执行

在特定的业务场景中：用户注册成功之后，发送短息通知用户（A服务为用户注册，B服务发送短信）A服务在完成用户注册之后（代码1），调用B服务发送短信，A服务完成B服务调用之后无需等待B服务的执行接口，直接执行提示用户注册成功（代码2），在这种需求下A服务调用B服务如果使用同步调用，必然降低A服务的执行效率，因此在这种场景下A服务需要通过**异步调用**调用B服务

异步调用：

当A服务调用B服务之后，无需等待B的调用结果，可以继续往下执行；那么服务间的异步通信该如何实现呢？

服务之间可以通过消息队列实现异步调用

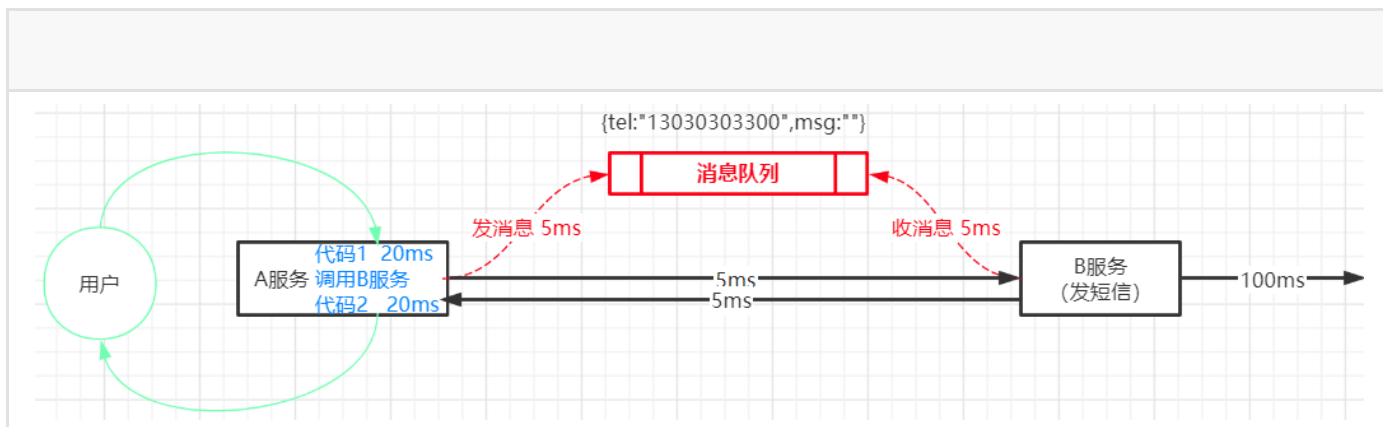
● 同步调用

- A服务调用B服务，需要等待B服务执行完毕的返回值，A服务才可以继续往下执行
- 同步调用可以通过REST和RPC完成
 - REST: ribbon、Feign
 - RPC: Dubbo

● 异步调用

- A服务调用B服务，而无需等待B服务的执行结果，也就是说在B服务执行的同时A服务可以继续往下执行
- 通过消息队列实现异步调用

1.2 消息队列概念



- MQ全称为Message Queue,消息队列（MQ）是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。
- 消息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信，直接调用通常是用于诸如远程过程调用的技术。

1.3 常用的消息队列产品

	RabbitMQ	ActiveMQ	RocketMQ	Kafka
所属社区/公司	Mozilla Public License	Apache	Ali	Apache
成熟度	成熟	成熟	比较成熟	成熟
授权方式	开源	开源	开源	开源
开发语言	Erlang	Java	Java	Scala&Java
客户端支持语言	官方支持Erlang, Java, Ruby等, 社区产出多种语言API, 几乎支持所有常用语言	Java、C、C++、Python、PHP、Perl、.net 等	Java C++ (不成熟)	官方支持Java, 开源社区有多语言版本, 如PHP, Python, Go, C/C++, Ruby, NodeJS等编程语言, 详见Kafka 客户端列表
协议支持	多协议支持:AMQP, XMPP, SMTP, STOMP	OpenWire、STOMP、REST、XMPP、AMQP	自己定义的一套(社区提供JMS--不成熟)	自有协议, 社区封装了HTTP协议支持
消息批量操作	不支持	支持	支持	支持
消息推拉模式	多协议, Pull/Push均有支持	多协议, Pull/Push均有支持	多协议, Pull/Push均有支持	Pull
HA	master/slave模式, master提供服务, slave仅作备份	基于ZooKeeper + LevelDB 的 Master-Slave 实现方式	支持多Master 模式、多 Master 多 Slave 模式, 异步复制模式、多 Master 多 Slave 模式, 同步双写	支持replica机制, leader宕掉后, 备份自动顶替, 并重新选举leader(基于Zookeeper)
数据可靠性	可以保证数据不丢, 有slave用作备份	master/slave	支持异步实时刷盘, 同步刷盘, 同步复制, 异步复制	数据可靠, 并且有replica 机制, 有容错容灾能力
单机吞吐量	其次(万级)	最差(万级)	最高(十万级)	次之(十万级)
消息延迟	微秒级	\	比kafka快	毫秒级
持久化能力	内存、文件, 支持数据堆积, 但数据堆积反过来影响生产速率	内存、文件、数据库	磁盘文件	磁盘文件, 只要磁盘容量够, 可以做到无限消息堆积
是否有序	若想有序, 只能使用一个Client	可以支持有序	有序	多Client保证有序
事务	不支持	支持	支持	不支持, 但可以通过Low Level API保证仅消费一次
集群	支持	支持	支持	支持
负载均衡	支持	支持	支持	支持
管理界面	较好	一般	命令行界面	官方只提供了命令行版, Yahoo开源自己的Kafka Web管理界面Kafka-Manager
部署方式	独立	独立	独立	独立

- RabbitMQ 稳定可靠,数据一致,支持多协议,有消息确认,基于erlang语言
- Kafka 高吞吐,高性能,快速持久化,无消息确认,无消息遗漏,可能会有有重复消息,依赖于zookeeper,成本高.
- ActiveMQ 不够灵活轻巧,对队列较多情况支持不好.
- RocketMQ 性能好,高吞吐,高可用性,支持大规模分布式, 协议支持单一

二、RabbitMQ

2.1 RabbitMQ介绍

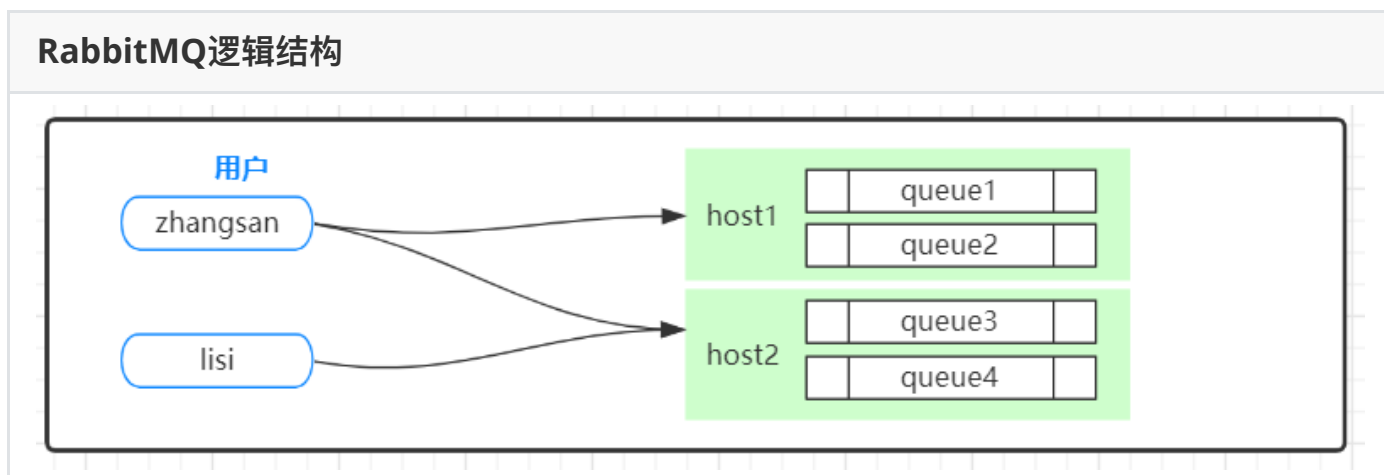
- RabbitMQ是一个在AMQP基础上完成的, 可复用的企业消息系统。他遵循Mozilla Public License开源协议。

- AMQP，即Advanced Message Queuing Protocol, 一个提供统一消息服务的应用层标准高级消息队列协议,是应用层协议的一个开放标准,为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息，并不受客户端/中间件不同产品，不同的开发语言等条件的限制。Erlang中的实现有 RabbitMQ 等。
- 主要特性：
 - 保证可靠性：使用一些机制来保证可靠性，如持久化、传输确认、发布确认
 - 灵活的路由功能
 - 可伸缩性：支持消息集群，多台RabbitMQ服务器可以组成一个集群
 - 高可用性：RabbitMQ集群中的某个节点出现问题时队列仍然可用
 - 支持多种协议
 - 支持多语言客户端
 - 提供良好的管理界面
 - 提供跟踪机制：如果消息出现异常，可以通过跟踪机制分析异常原因
 - 提供插件机制：可通过插件进行多方面扩展

2.2 RabbitMQ安装和配置

- 参考安装文档：RabbitMQ安装及配置.pdf
链接：<http://note.youdao.com/noteshare?id=b01d0f9ed0499610fc8f7b2fc76ef17e&sub=57DF1D9EB49C4529AA4EA38A517901B4>

2.3 RabbitMQ逻辑结构



三、RabbitMQ用户管理

RabbitMQ默认提供了一个guests账号，但是此账号不能用作远程登录，也就是不能在管理系统的登录；我们可以创建一个新的账号并授予响应的管理权限来实现远程登录

3.1 逻辑结构

- 用户
- 虚拟主机
- 队列

3.2 用户管理

3.2.1 命令行用户管理

- 在linux中使用命令行创建用户

```
1 ## 进入到rabbitmq的sbin目录
2 cd /usr/local/rabbitmq_server-3.7.0/sbin
3
4 ## 新增用户
5 ./rabbitmqctl add_user ytao admin123
```

- 设置用户级别

```
1 ## 用户级别:
2 ## 1.administrator 可以登录控制台、查看所有信息、可以对RabbitMQ进行管理
3 ## 2.monitoring 监控者 登录控制台、查看所有信息
4 ## 3.policymaker 策略制定者 登录控制台、指定策略
5 ## 4.managment 普通管理员 登录控制台
6
7 ./rabbitmqctl set_user_tags ytao administrator
```

3.2.2 管理系统进行用户管理

- 管理系统登录: 访问<http://47.96.11.185:15672/>

1. 新增用户

The screenshot shows the RabbitMQ Management interface at the URL `47.96.11.185:15672/#/users`. The 'Admin' tab is selected. The 'Add a user' form is visible with the following fields:

- Username: `zhangsan`
- Password: `*****` (confirm)
- Tags: `administrator` (selected)

Annotations on the image:

- 1. 点击admin (Click Admin)
- 2. 点击Users (Click Users)
- 3. 输入用户基本信息 (Enter user basic information)
- 4. 点击AddUser (Click Add User)
- 选择用户身份 (Select user identity) - points to the 'administrator' tag.

Name	Tags	Can access virtual hosts	Has password
ytao	administrator	No access	•

2. 创建虚拟主机

The screenshot shows the RabbitMQ Management interface at the URL `47.96.11.185:15672/#/vhosts`. The 'Admin' tab is selected. The 'Add a new virtual host' form is visible with the following fields:

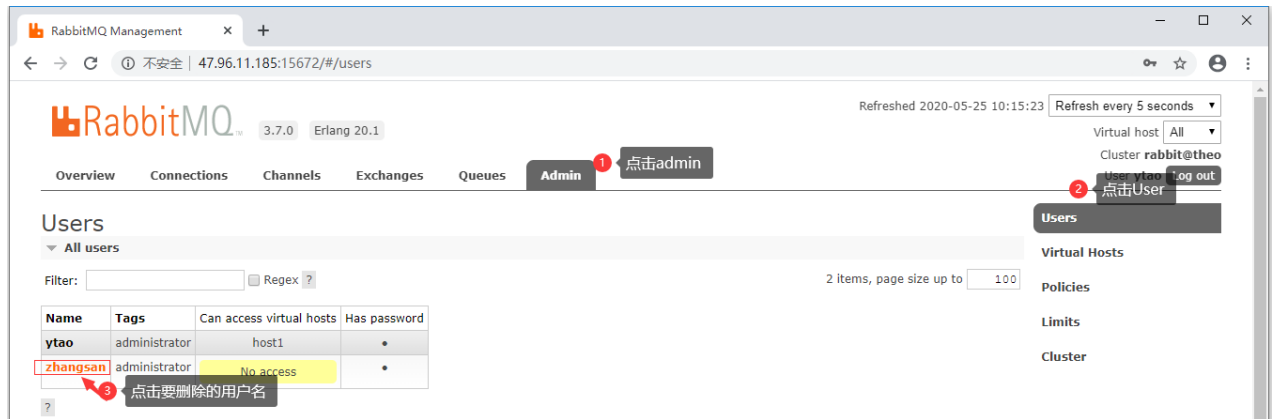
- Name: `host2`

Annotations on the image:

- 1. 点击admin (Click Admin)
- 2. 点击VH (Click VH)
- 3. 输入虚拟主机名称 (Enter virtual host name)
- 4. 点击Add (Click Add)

Overview			Messages			Network		Message rates	
Name	Users	State	Ready	Unacked	Total	From client	To client	publish	deliver / get
host1	ytao	running	NaN	NaN	NaN				

3. 删除用户



User: zhangsan

This user does not have permission to access any virtual hosts.
Use "Set Permission" below to grant permission to access virtual hosts.

Overview

Tags
administrator

Can log in with password: •

► Permissions

► Topic permissions

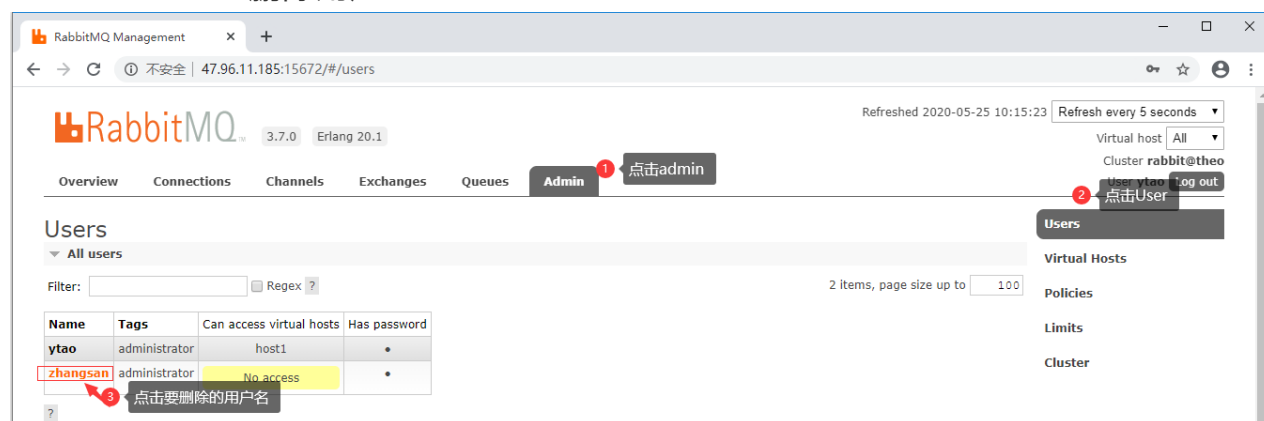
► Update this user

Delete this user

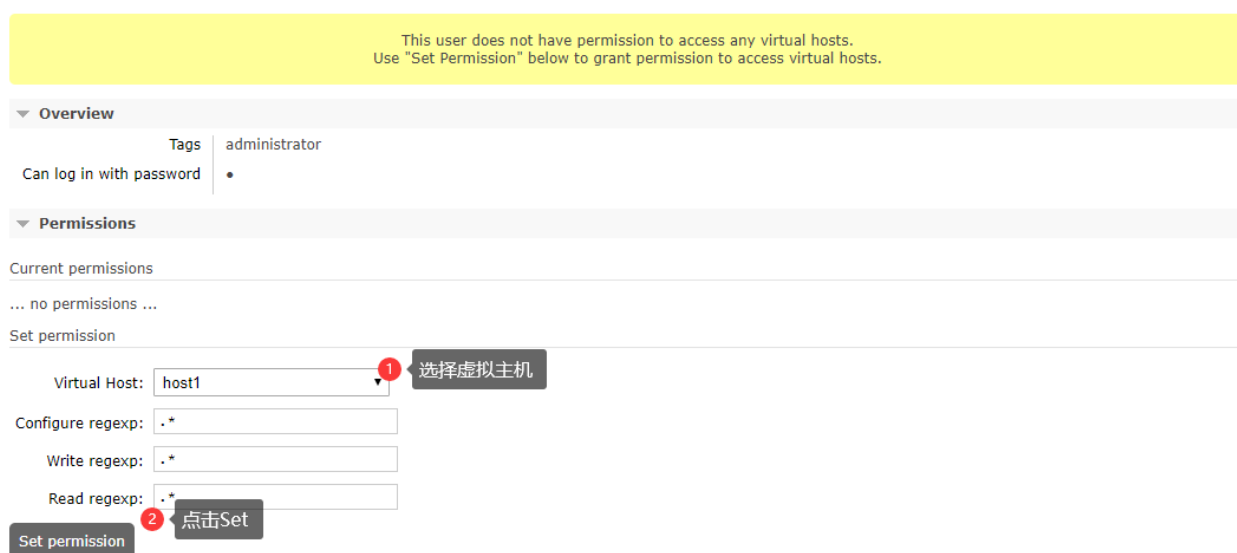
Delete ④ 点击 "delete"

4. 用户绑定虚拟主机

More Actions 3. 删除用户



User: zhangsan



四、RabbitMQ 工作方式

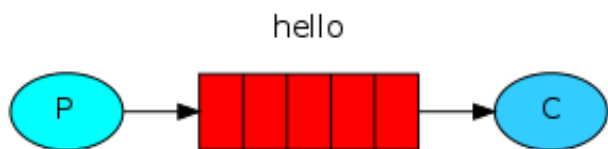
RabbitMQ 提供了多种消息的通信方式—工作模式

<https://www.rabbitmq.com/getstarted.html>

消息通信是由两个角色完成：消息生产者（producer）和 消息消费者（Consumer）

4.1 简单模式

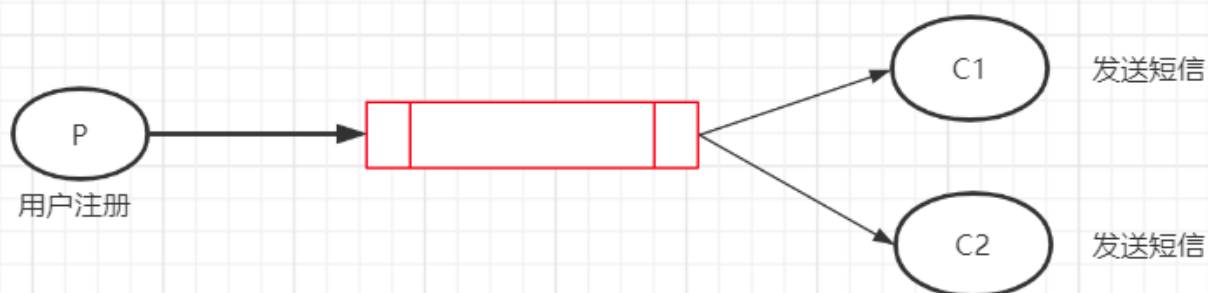
一个队列只有一个消费者



生产者将消息发送到队列，消费者从队列取出数据

4.2 工作模式

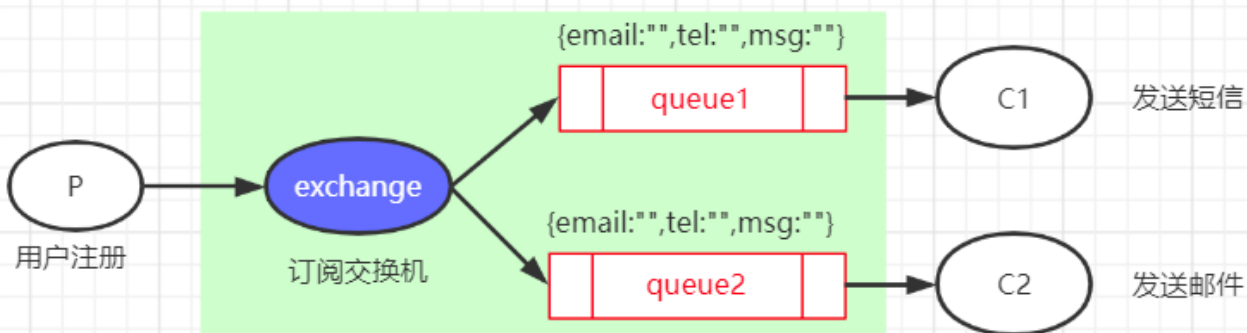
多个消费者监听同一个队列



多个消费者监听同一个队列，但多个消费者中只有一个消费者会成功的消费消息

4.3 订阅模式

一个交换机绑定多个消息队列，每个消息队列有一个消费者监听

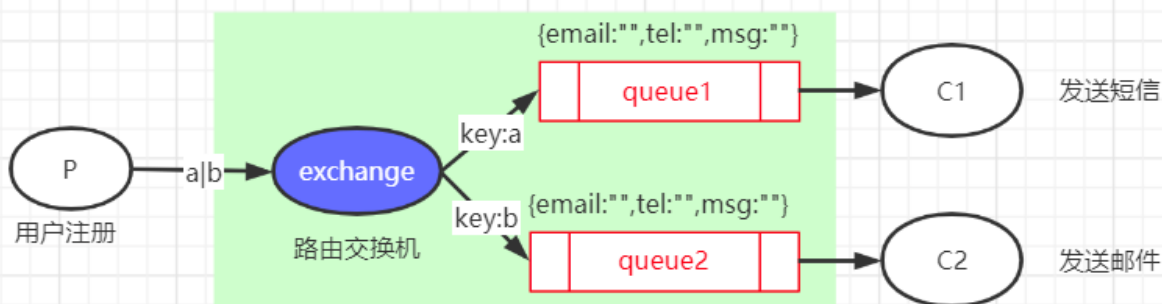


消息生产者发送的消息可以被每一个消费者接收

4.4 路由模式

一个交换机绑定多个消息队列，每个消息队列都由自己唯一的key，每个消息队列有一个消费者监听

路由模式



五、RabbitMQ交换机和队列管理

5.1 创建队列

RabbitMQ 3.7.0 Erlang 20.1

Overview Connections Channels Exchanges **Queues** 1 点击queues

Queues

▼ All queues (0)

Pagination

Page 1 of 0 - Filter: ☐ Regex ?

... no queues ...

▼ Add a new queue

Virtual host: 2 选择目标虚拟主机（队列必须在虚拟主机中）

Name: 3 输入队列名

Durability:

Auto delete: ?

Arguments: = String ▼ 4 设置队列参数

Add Message TTL ? | Auto expire ? | Max length ? | Max length bytes ? | Overflow behaviour ?

Dead letter exchange ? | Dead letter routing key ? | Maximum priority ?

Lazy mode ? Master locator ?

Add queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

5.2 创建交换机

RabbitMQ 3.7.0 Erlang 20.1

Overview Connections Channels **Exchanges** Queues Admin

Virtual host	Name	Type	Durable	Internal	Auto delete
host2	amq.rabbitmq.trace	topic	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
host2	amq.topic	topic	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

▼ Add a new exchange

Virtual host: **2 选择虚拟主机**

Name: **3 输入交换机名称**

Type: **4 选择交换机类型 fanout 订阅交换机 redirect 路由交换机**

Durability:

Auto delete:


Internal:

Arguments: =

Add **Alternate exchange** ?

5 点击Add Add exchange

5.3 交换机绑定队列

 3.7.0 Erlang 20.1

Overview

Connections

Channels

Exchanges

Queues

Admin

host1	amq.match	headers	D		
host1	amq.rabbitmq.trace	topic	D I		
host1	amq.topic	topic	D		
host1	ex1	fanout	D		
host1	ex2	direct	D		

Exchange: ex1 in virtual host host1

Overview

Message rates last minute ?

Currently idle

Details

Type	fanout
Features	durable: true
Policy	

Bindings

This exchange

↓

To	Routing key	Arguments	
queue3			Unbind

Add binding from this exchange

To queue ▼

: queue4 1 绑定的队列名称

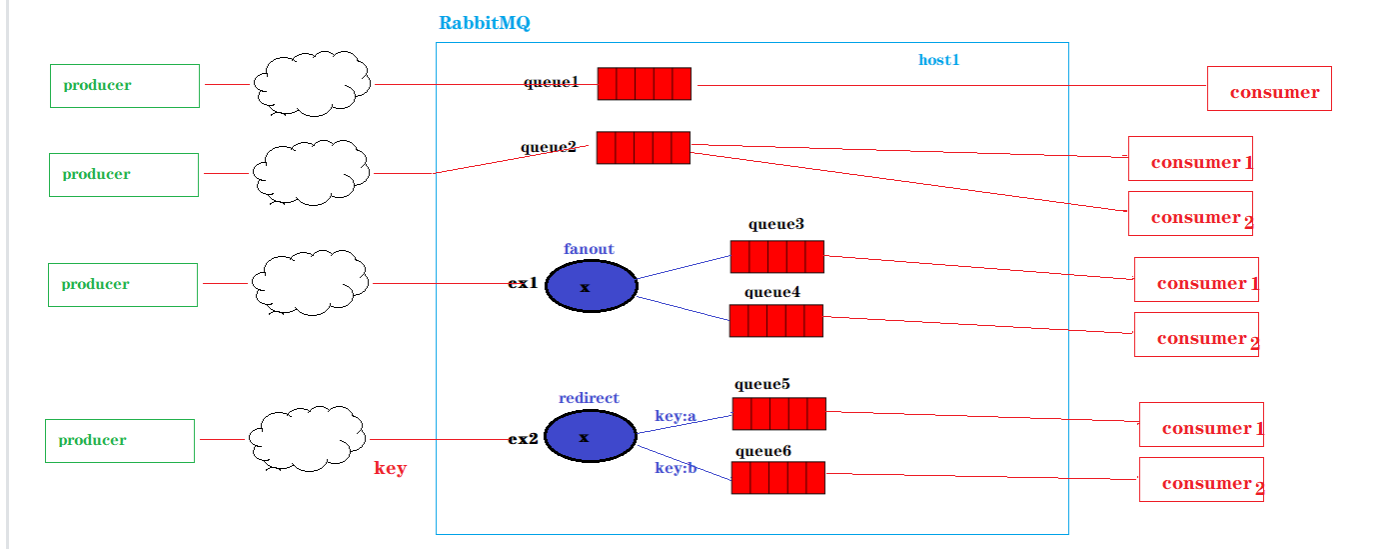
Routing key: 2 如果是路由交换机则需要指定key

Arguments: = String ▼

Bind

六、在普通的Maven应用中使用MQ

RabbitMQ 队列结构



6.1 简单模式

6.1.1 消息生产者

- 创建Maven项目
- 添加RabbitMQ连接所需要的依赖

```

1  <!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
2  <dependency>
3      <groupId>com.rabbitmq</groupId>
4      <artifactId>amqp-client</artifactId>
5      <version>4.10.0</version>
6  </dependency>
7  <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->
8  <dependency>
9      <groupId>org.slf4j</groupId>
10     <artifactId>slf4j-log4j12</artifactId>
11     <version>1.7.25</version>
12     <scope>test</scope>
13 </dependency>
14 <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
15 <dependency>
16     <groupId>org.apache.commons</groupId>
17     <artifactId>commons-lang3</artifactId>
18     <version>3.9</version>

```

```
19 </dependency>
```

- 在resources目录下创建log4j.properties

```
1 log4j.rootLogger=DEBUG,A1 log4j.logger.com.taotao = DEBUG
2 log4j.logger.org.mybatis = DEBUG
3 log4j.appender.A1=org.apache.log4j.ConsoleAppender
4 log4j.appender.A1.layout=org.apache.log4j.PatternLayout
5 log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd
HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n
```

- 创建MQ连接帮助类

```
1 package com.qfedu.mq.utils;
2
3 import com.rabbitmq.client.Connection;
4 import com.rabbitmq.client.ConnectionFactory;
5
6 import java.io.IOException;
7 import java.util.concurrent.TimeoutException;
8
9 public class ConnectionUtil {
10
11     public static Connection getConnection() throws IOException,
12     TimeoutException {
13         //1.创建连接工厂
14         ConnectionFactory factory = new ConnectionFactory();
15         //2.在工厂对象中设置MQ的连接信息
16         (ip,port,virtualhost,username,password)
17         factory.setHost("47.96.11.185");
18         factory.setPort(5672);
19         factory.setVirtualHost("host1");
20         factory.setUsername("ytao");
21         factory.setPassword("admin123");
22         //3.通过工厂对象获取与MQ的连接
23         Connection connection = factory.newConnection();
24         return connection;
25     }
26 }
```

- 消息生产者发送消息

```
1 package com.qfedu.mq.service;
2
3 import com.qfedu.mq.utils.ConnectionUtil;
4 import com.rabbitmq.client.Channel;
5 import com.rabbitmq.client.Connection;
6
7 public class SendMsg {
8
9     public static void main(String[] args) throws Exception{
10
11         String msg = "Hello HuangDaoJun!";
12         Connection connection = ConnectionUtil.getConnection();
13         Channel channel = connection.createChannel();
14
15         //定义队列(使用Java代码在MQ中新建一个队列)
16         //参数1: 定义的队列名称
17         //参数2: 队列中的数据是否持久化(如果选择了持久化)
18         //参数3: 是否排外(当前队列是否为当前连接私有)
19         //参数4: 自动删除(当此队列的连接数为0时, 此队列会销毁(无论队列中是否
20         //还有数据))
21         //参数5: 设置当前队列的参数
22         //channel.queueDeclare("queue7", false, false, false, null);
23
24         //参数1: 交换机名称, 如果直接发送信息到队列, 则交换机名称为""
25         //参数2: 目标队列名称
26         //参数3: 设置当前这条消息的属性(设置过期时间 10)
27         //参数4: 消息的内容
28         channel.basicPublish("", "queue1", null, msg.getBytes());
29         System.out.println("发送: " + msg);
30
31         channel.close();
32         connection.close();
33     }
34 }
```

6.1.2 消息消费者

- 创建Maven项目
- 添加依赖
- log4j.properties

- ConnetionUtil.java
- 消费者消费消息

```
1 package com.qfedu.mq.service;
2
3 import com.qfedu.mq.utils.ConnectionUtil;
4 import com.rabbitmq.client.*;
5
6 import java.io.IOException;
7 import java.util.concurrent.TimeoutException;
8
9 public class ReceiveMsg {
10
11     public static void main(String[] args) throws IOException,
12     TimeoutException {
13         Connection connection = ConnectionUtil.getConnection();
14         Channel channel = connection.createChannel();
15
16         Consumer consumer = new DefaultConsumer(channel){
17             @Override
18             public void handleDelivery(String consumerTag, Envelope
19             envelope,
20                                     AMQP.BasicProperties properties, byte[]
21             body) throws IOException {
22                 //body就是从队列中获取的数据
23                 String msg = new String(body);
24                 System.out.println("接收: "+msg);
25             }
26         };
27
28         channel.basicConsume("queue1", true, consumer);
29     }
30 }
```

6.2 工作模式

一个发送者多个消费者

6.2.1 发送者

```

1 public class SendMsg {
2
3     public static void main(String[] args) throws Exception{
4         System.out.println("请输入消息: ");
5         Scanner scanner = new Scanner(System.in);
6         String msg = null;
7         while(!"quit".equals(msg = scanner.nextLine())){
8             Connection connection = ConnectionUtil.getConnection();
9             Channel channel = connection.createChannel();
10
11             channel.basicPublish("", "queue2", null, msg.getBytes());
12             System.out.println("发送: " + msg);
13
14             channel.close();
15             connection.close();
16         }
17     }
18
19 }

```

6.2.2 消费者1

```

1 public class ReceiveMsg {
2
3     public static void main(String[] args) throws Exception {
4         Connection connection = ConnectionUtil.getConnection();
5         Channel channel = connection.createChannel();
6
7         Consumer consumer = new DefaultConsumer(channel){
8             @Override
9             public void handleDelivery(String consumerTag, Envelope
10 envelope, AMQP.BasicProperties properties, byte[] body) throws
11 IOException {
12                 //body就是从队列中获取的数据
13                 String msg = new String(body);
14                 System.out.println("Consumer1接收: "+msg);
15                 if("wait".equals(msg)){
16                     try {
17                         Thread.sleep(10000);
18                     } catch (InterruptedException e) {
19

```

```

17         e.printStackTrace();
18     }
19 }
20 }
21 };
22
23     channel.basicConsume("queue2", true, consumer);
24 }
25 }

```

6.2.3 消费者2

```

1 public class ReceiveMsg {
2
3     public static void main(String[] args) throws IOException,
4     TimeoutException {
5         Connection connection = ConnectionUtil.getConnection();
6         Channel channel = connection.createChannel();
7
8         Consumer consumer = new DefaultConsumer(channel){
9             @Override
10             public void handleDelivery(String consumerTag, Envelope
11             envelope, AMQP.BasicProperties properties, byte[] body) throws
12             IOException {
13                 //body就是从队列中获取的数据
14                 String msg = new String(body);
15                 System.out.println("Consumer2接收: "+msg);
16             }
17         };
18
19         channel.basicConsume("queue2", true, consumer);
20     }
21 }

```

6.3 订阅模式

6.3.1 发送者 发送消息到交换机

```

1 public class SendMsg {
2
3     public static void main(String[] args) throws Exception{
4         System.out.println("请输入消息: ");
5     }
6 }

```

```

5      Scanner scanner = new Scanner(System.in);
6      String msg = null;
7      while(!"quit".equals(msg = scanner.nextLine())){
8          Connection connection = ConnectionUtil.getConnection();
9          Channel channel = connection.createChannel();
10
11          channel.basicPublish("ex1", "", null, msg.getBytes());
12          System.out.println("发送: " + msg);
13
14          channel.close();
15          connection.close();
16      }
17  }
18
19 }

```

6.3.2 消费者1

```

1  public class ReceiveMsg1 {
2
3      public static void main(String[] args) throws Exception {
4          Connection connection = ConnectionUtil.getConnection();
5          Channel channel = connection.createChannel();
6
7          Consumer consumer = new DefaultConsumer(channel){
8              @Override
9              public void handleDelivery(String consumerTag, Envelope
10 envelope, AMQP.BasicProperties properties, byte[] body) throws
11 IOException {
12
13              //body就是从队列中获取的数据
14              String msg = new String(body);
15              System.out.println("Consumer1接收: "+msg);
16              if("wait".equals(msg)){
17                  try {
18                      Thread.sleep(10000);
19                  } catch (InterruptedException e) {
20                      e.printStackTrace();
21                  }
22              }
23          }
24      };
25  }

```

```

23         channel.basicConsume("queue3",true,consumer);
24     }
25 }

```

6.3.3 消费者2

```

1  public class ReceiveMsg2 {
2
3      public static void main(String[] args) throws IOException,
4      TimeoutException {
5          Connection connection = ConnectionUtil.getConnection();
6          Channel channel = connection.createChannel();
7
8          Consumer consumer = new DefaultConsumer(channel){
9              @Override
10             public void handleDelivery(String consumerTag, Envelope
11             envelope, AMQP.BasicProperties properties, byte[] body) throws
12             IOException {
13                 //body就是从队列中获取的数据
14                 String msg = new String(body);
15                 System.out.println("Consumer2接收: "+msg);
16             }
17         };
18
19         channel.basicConsume("queue4",true,consumer);
20     }
21 }

```

6.4 路由模式

6.4.1 发送者 发送消息到交换机

```

1  public class SendMsg {
2
3      public static void main(String[] args) throws Exception{
4          System.out.println("请输入消息: ");
5          Scanner scanner = new Scanner(System.in);
6          String msg = null;
7          while(!"quit".equals(msg = scanner.nextLine())){
8              Connection connection = ConnectionUtil.getConnection();
9              Channel channel = connection.createChannel();
10

```

```

11         if(msg.startsWith("a")){
12             channel.basicPublish("ex2", "a", null, msg.getBytes());
13         }else if(msg.startsWith("b")){
14             channel.basicPublish("ex2", "b", null, msg.getBytes());
15         }
16         System.out.println("发送: " + msg);
17
18         channel.close();
19         connection.close();
20     }
21 }
22
23 }

```

6.4.2 消费者1

```

1 public class ReceiveMsg1 {
2
3     public static void main(String[] args) throws Exception {
4         Connection connection = ConnectionUtil.getConnection();
5         Channel channel = connection.createChannel();
6
7         Consumer consumer = new DefaultConsumer(channel){
8             @Override
9             public void handleDelivery(String consumerTag, Envelope
10 envelope, AMQP.BasicProperties properties, byte[] body) throws
11 IOException {
12
13                 //body就是从队列中获取的数据
14                 String msg = new String(body);
15                 System.out.println("Consumer1接收: "+msg);
16                 if("wait".equals(msg)){
17                     try {
18                         Thread.sleep(10000);
19                     } catch (InterruptedException e) {
20                         e.printStackTrace();
21                     }
22                 }
23             };
24
25         channel.basicConsume("queue5", true, consumer);
26     }
27 }

```

25 }

6.4.3 消费者2

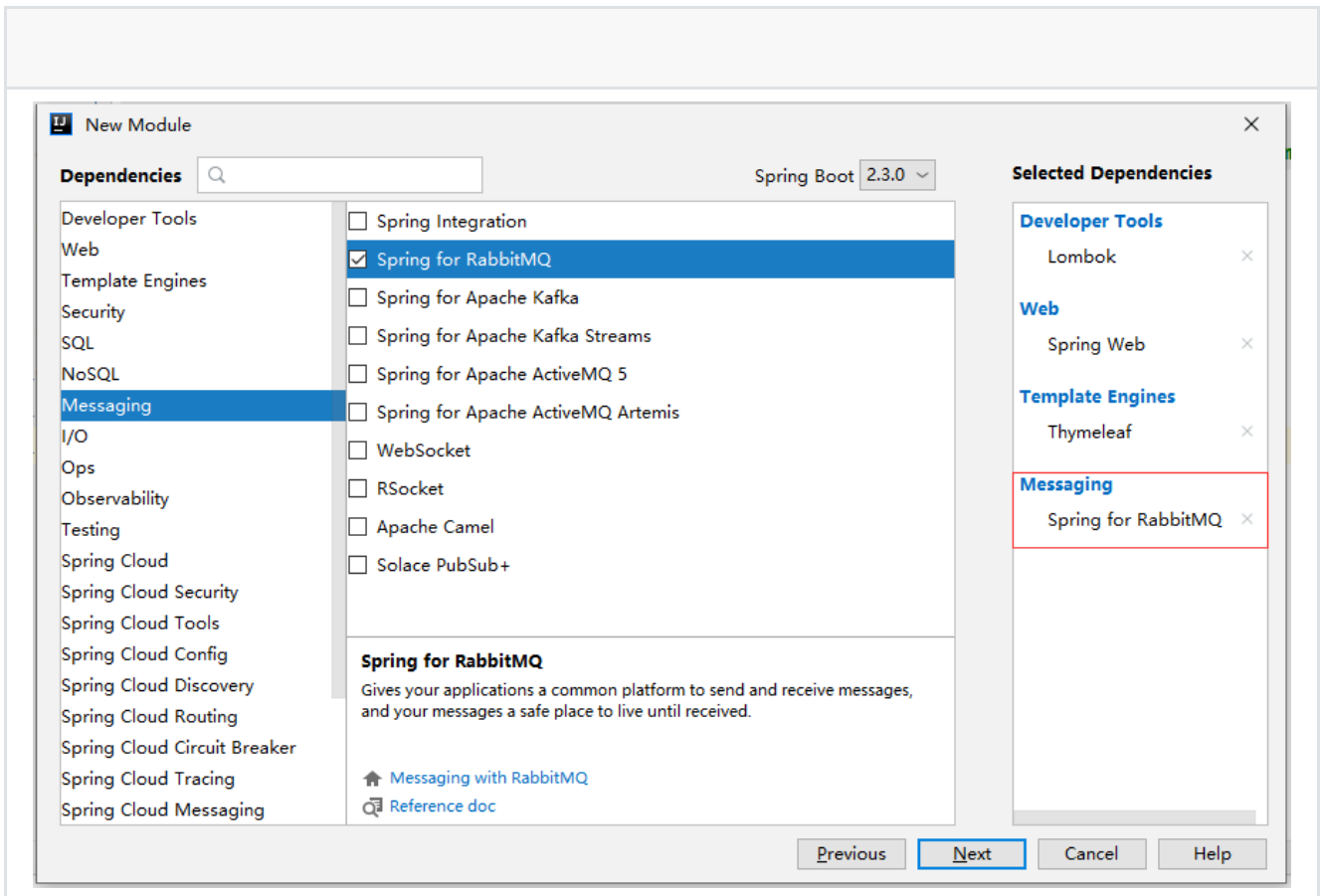
```
1 public class ReceiveMsg2 {
2
3     public static void main(String[] args) throws IOException,
4     TimeoutException {
5         Connection connection = ConnectionUtil.getConnection();
6         Channel channel = connection.createChannel();
7
8         Consumer consumer = new DefaultConsumer(channel){
9             @Override
10             public void handleDelivery(String consumerTag, Envelope
11             envelope, AMQP.BasicProperties properties, byte[] body) throws
12             IOException {
13                 //body就是从队列中获取的数据
14                 String msg = new String(body);
15                 System.out.println("Consumer2接收: "+msg);
16             }
17         };
18
19         channel.basicConsume("queue6", true, consumer);
20     }
21 }
```

七、在SpringBoot应用中使用MQ

SpringBoot应用可以完成自动配置及依赖注入——可以通过Spring直接提供与MQ的连接对象

7.1 消息生产者

- 创建SpringBoot应用，添加依赖



- 配置 application.yml

```
1  server:
2    port: 9001
3  spring:
4    application:
5      name: producer
6    rabbitmq:
7      host: 47.96.11.185
8      port: 5672
9      virtual-host: host1
10     username: ytao
11     password: admin123
```

- 发送消息

```
1  @Service
2  public class TestService {
3
4      @Resource
5      private AmqpTemplate amqpTemplate;
6  }
```

```
7     public void sendMsg(String msg){
8
9         //1. 发送消息到队列
10        amqpTemplate.convertAndSend("queue1",msg);
11
12        //2. 发送消息到交换机(订阅交换机)
13        amqpTemplate.convertAndSend("ex1","",msg);
14
15        //3. 发送消息到交换机(路由交换机)
16        amqpTemplate.convertAndSend("ex2","a",msg);
17
18    }
19
20 }
```

7.2 消息消费者

- 创建项目添加依赖
- 配置yml
- 接收消息

```
1  @Service
2  //@RabbitListener(queues = {"queue1","queue2"})
3  @RabbitListener(queues = "queue1")
4  public class ReceiveMsgService {
5
6      @RabbitHandler
7      public void receiveMsg(String msg){
8          System.out.println("接收MSG: "+msg);
9      }
10
11 }
```

八、使用RabbitMQ传递对象

RabbitMQ是消息队列，发送和接收的都是字符串/字节数组类型的消息

8.1 使用序列化对象

要求:

- 传递的对象实现序列化接口
- 传递的对象的包名、类名、属性名必须一致
- 消息提供者

```
1  @Service
2  public class MQService {
3
4      @Resource
5      private AmqpTemplate amqpTemplate;
6
7      public void sendGoodsToMq(Goods goods) {
8          //消息队列可以发送 字符串、字节数组、序列化对象
9          amqpTemplate.convertAndSend("", "queue1", goods);
10     }
11
12 }
```

- 消息消费者

```
1  @Component
2  @RabbitListener(queues = "queue1")
3  public class ReceiveService {
4
5      @RabbitHandler
6      public void receiveMsg(Goods goods) {
7          System.out.println("Goods---"+goods);
8      }
9
10 }
```

8.2 使用序列化字节数组

要求:

- 传递的对象实现序列化接口
- 传递的对象的包名、类名、属性名必须一致
- 消息提供者

```

1  @Service
2  public class MQService {
3
4      @Resource
5      private AmqpTemplate amqpTemplate;
6
7      public void sendGoodsToMq(Goods goods){
8          //消息队列可以发送 字符串、字节数组、序列化对象
9          byte[] bytes = SerializationUtils.serialize(goods);
10         amqpTemplate.convertAndSend("", "queue1", bytes);
11     }
12
13 }

```

- 消息消费者

```

1  @Component
2  @RabbitListener(queues = "queue1")
3  public class ReceiveService {
4
5      @RabbitHandler
6      public void receiveMsg(byte[] bs){
7          Goods goods = (Goods) SerializationUtils.deserialize(bs);
8          System.out.println("byte[]---"+goods);
9      }
10
11 }

```

8.3 使用JSON字符串传递

要求：对象的属性名一直

- 消息提供者

```

1  @Service
2  public class MQService {
3
4      @Resource
5      private AmqpTemplate amqpTemplate;
6
7      public void sendGoodsToMq(Goods goods) throws
      JsonProcessingException {

```

```

8      //消息队列可以发送 字符串、字节数组、序列化对象
9      ObjectMapper objectMapper = new ObjectMapper();
10     String msg = objectMapper.writeValueAsString(goods);
11     amqpTemplate.convertAndSend("", "queue1", msg);
12 }
13
14 }
```

- 消息消费者

```

1  @Component
2  @RabbitListener(queues = "queue1")
3  public class ReceiveService {
4
5      @RabbitHandler
6      public void receiveMsg(String msg) throws
7      JsonProcessingException {
8          ObjectMapper objectMapper = new ObjectMapper();
9          Goods goods = objectMapper.readValue(msg, Goods.class);
10         System.out.println("String---"+msg);
11     }
12 }
```

九、基于Java的交换机与队列创建

我们使用消息队列，消息队列和交换机可以通过管理系统完成创建，也可以在应用程序中通过Java代码来完成创建

9.1 普通Maven项目交换机及队列创建

- 使用Java代码新建队列

```

1  //1.定义队列 （使用Java代码在MQ中新建一个队列）
2  //参数1： 定义的队列名称
3  //参数2： 队列中的数据是否持久化（如果选择了持久化）
4  //参数3： 是否排外（当前队列是否为当前连接私有）
5  //参数4： 自动删除（当此队列的连接数为0时，此队列会销毁（无论队列中是否还有数据））
6  //参数5： 设置当前队列的参数
7  channel.queueDeclare("queue7", false, false, false, null);
```

- 新建交换机

```

1 //定义一个“订阅交换机”
2 channel.exchangeDeclare("ex3", BuiltinExchangeType.FANOUT);
3 //定义一个“路由交换机”
4 channel.exchangeDeclare("ex4", BuiltinExchangeType.DIRECT);

```

- 绑定队列到交换机

```

1 //绑定队列
2 //参数1: 队列名称
3 //参数2: 目标交换机
4 //参数3: 如果绑定订阅交换机参数为"",如果绑定路由交换机则表示设置队列的key
5 channel.queueBind("queue7", "ex4", "k1");
6 channel.queueBind("queue8", "ex4", "k2");

```

9.2 SpringBoot应用中通过配置完成队列的创建

```

1 @Configuration
2 public class RabbitMQConfiguration {
3
4     //声明队列
5     @Bean
6     public Queue queue9(){
7         Queue queue9 = new Queue("queue9");
8         //设置队列属性
9         return queue9;
10    }
11    @Bean
12    public Queue queue10(){
13        Queue queue10 = new Queue("queue10");
14        //设置队列属性
15        return queue10;
16    }
17
18    //声明订阅模式交换机
19    @Bean
20    public FanoutExchange ex5(){
21        return new FanoutExchange("ex5");
22    }
23
24    //声明路由模式交换机

```

```

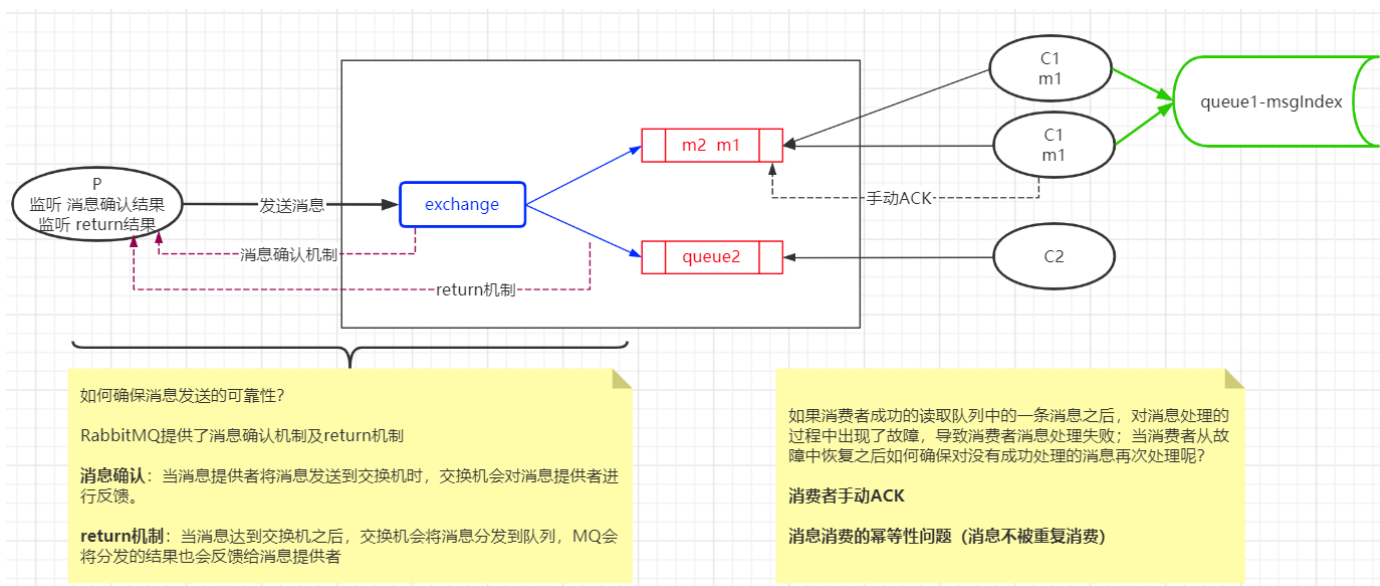
25     @Bean
26     public DirectExchange ex6(){
27         return new DirectExchange("ex6");
28     }
29
30     //绑定队列
31     @Bean
32     public Binding bindingQueue9(Queue queue9, DirectExchange ex6){
33         return BindingBuilder.bind(queue9).to(ex6).with("k1");
34     }
35     @Bean
36     public Binding bindingQueue10(Queue queue10, DirectExchange ex6){
37         return BindingBuilder.bind(queue10).to(ex6).with("k2");
38     }
39 }
40

```

十、消息的可靠性

消息的可靠性：从生产者发送消息——消息队列存储消息——消费者消费消息的整个过程中消息的安全性及可控性。

- 生产者
- 消息队列
- 消费者

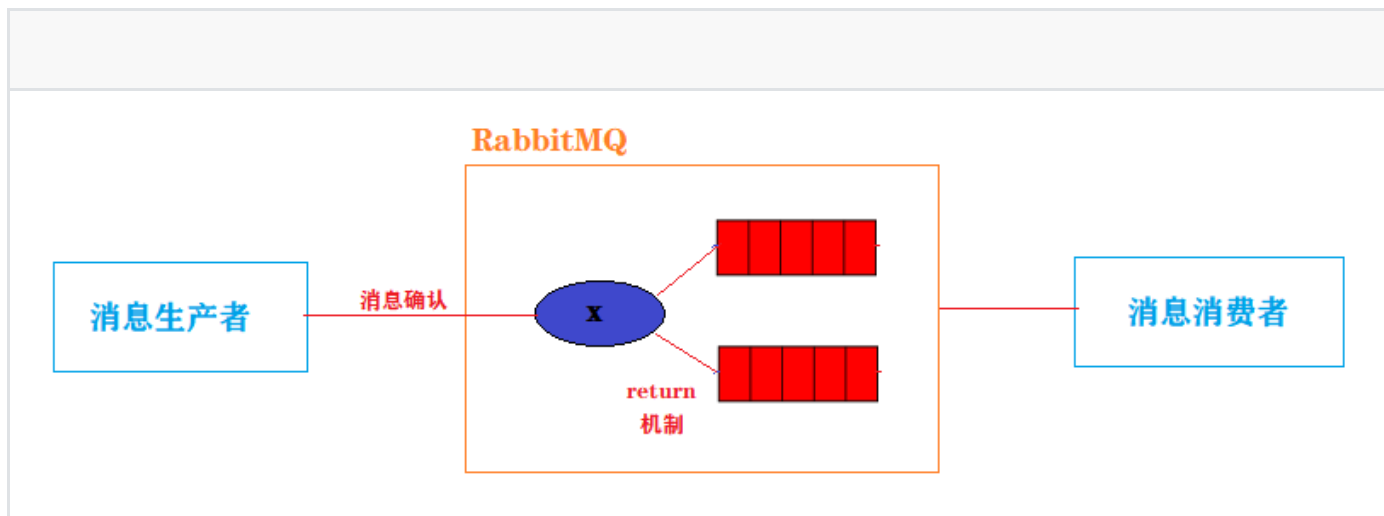


10.1 RabbitMQ事务

RabbitMQ事务指的是基于客户端实现的事务管理，当在消息发送过程中添加了事务，处理效率降低几十倍甚至上百倍

```
1 Connection connection = RabbitMQUtil.getConnection(); //connection 表示与 host1的连接
2 Channel channel = connection.createChannel();
3 channel.txSelect(); //开启事务
4 try{
5     channel.basicPublish("ex4", "k1", null, msg.getBytes());
6     channel.txCommit(); //提交事务
7 }catch (Exception e){
8     channel.txRollback(); //事务回滚
9 }finally{
10     channel.close();
11     connection.close();
12 }
```

10.2 RabbitMQ消息确认和return机制



消息确认机制：确认消息提供者是否成功发送消息到交换机

return机制：确认消息是否成功的从交换机分发到队列

10.2.1 普通Maven项目的消息确认

- 普通confirm方式

```

1 //1.发送消息之前开启消息确认
2 channel.confirmSelect();
3
4 channel.basicPublish("ex1", "a", null, msg.getBytes());
5
6 //2.接收消息确认
7 boolean b = channel.waitForConfirms();
8
9 System.out.println("发送: " + (b?"成功":"失败"));

```

● 批量confirm方式

```

1 //1.发送消息之前开启消息确认
2 channel.confirmSelect();
3
4 //2.批量发送消息
5 for (int i=0 ; i<10 ; i++){
6     channel.basicPublish("ex1", "a", null, msg.getBytes());
7 }
8
9 //3.接收批量消息确认: 发送的所有消息中, 如果有一条是失败的, 则所有消息发送直接失败,
    抛出IO异常
10 boolean b = channel.waitForConfirms();

```

● 异步confirm方式

```

1 //发送消息之前开启消息确认
2 channel.confirmSelect();
3
4 //批量发送消息
5 for (int i=0 ; i<10 ; i++){
6     channel.basicPublish("ex1", "a", null, msg.getBytes());
7 }
8
9 //假如发送消息需要10s, waitForConfirms会进入阻塞状态
10 //boolean b = channel.waitForConfirms();
11
12 //使用监听器异步confirm
13 channel.addConfirmListener(new ConfirmListener() {
14     //参数1: long l 返回消息的表示
15     //参数2: boolean b 是否为批量confirm
16     public void handleAck(long l, boolean b) throws IOException {

```

```

17         System.out.println("~~~~~消息成功发送到交换机");
18     }
19     public void handleNack(long l, boolean b) throws IOException {
20         System.out.println("~~~~~消息发送到交换机失败");
21     }
22 });

```

10.2.2 普通Maven项目的return机制

- 添加return监听器
- 发送消息是指定第三个参数为true
- 由于监听器监听是异步处理，所以在消息发送之后不能关闭channel

```

1  String msg = "Hello HuangDaoJun!";
2  Connection connection = ConnectionUtil.getConnection();           //相当于JDBC
   操作的数据库连接
3  Channel channel = connection.createChannel();                     //相当于JDBC
   操作的statement
4
5  //return机制：监控交换机是否将消息分发到队列
6  channel.addReturnListener(new ReturnListener() {
7      public void handleReturn(int i, String s, String s1, String
   s2,AMQP.BasicProperties basicProperties,byte[] bytes) throws
   IOException {
8          //如果交换机分发消息到队列失败，则会执行此方法（用来处理交换机分发消息到队
   列失败的情况）
9          System.out.println("*****"+i); //标识
10         System.out.println("*****"+s); //
11         System.out.println("*****"+s1); //交换机名
12         System.out.println("*****"+s2); //交换机对应的队列的key
13         System.out.println("*****"+new String(bytes)); //发送的消息
14     }
15 });
16
17 //发送消息
18 //channel.basicPublish("ex2", "c", null, msg.getBytes());
19 channel.basicPublish("ex2", "c", true, null, msg.getBytes());

```


10.3 在SpringBoot应用实现消息确认与return监听

10.3.1 配置application.yml,开启消息确认和return监听

```
1 spring:
2   rabbitmq:
3     publisher-confirm-type: simple  ## 开启消息确认模式
4     publisher-returns: true      ##使用return监听机制
```

10.3.2 创建confirm和return监听

- 消息确认

```
1 @Component
2 public class MyConfirmListener implements
RabbitTemplate.ConfirmCallback {
3
4     @Autowired
5     private AmqpTemplate amqpTemplate;
6
7     @Autowired
8     private RabbitTemplate rabbitTemplate;
9
10    @PostConstruct
11    public void init(){
12        rabbitTemplate.setConfirmCallback(this);
13    }
14
15    @Override
16    public void confirm(CorrelationData correlationData, boolean b,
String s) {
17        //参数b 表示消息确认结果
18        //参数s 表示发送的消息
19        if(b){
20            System.out.println("消息发送到交换机成功!");
21        }else{
22            System.out.println("消息发送到交换机失败!");
23            amqpTemplate.convertAndSend("ex4", "", s);
24        }
25    }
26
27 }
```

• return 机制

```

1  @Component
2  public class MyReturnListener implements RabbitTemplate.ReturnsCallback
3  {
4      @Autowired
5      private AmqpTemplate amqpTemplate;
6
7      @Autowired
8      private RabbitTemplate rabbitTemplate;
9
10     @PostConstruct
11     public void init(){
12         rabbitTemplate.setReturnsCallback(this);
13     }
14
15     @Override
16     public void returnedMessage(ReturnedMessage returnedMessage) {
17         System.out.println("消息从交换机分发到队列失败");
18         String exchange = returnedMessage.getExchange();
19         String routingKey = returnedMessage.getRoutingKey();
20         String msg = returnedMessage.getMessage().toString();
21         amqpTemplate.convertAndSend(exchange,routingKey,msg);
22     }
23 }

```

10.4 RabbitMQ 消费者手动应答

```

1  @Component
2  @RabbitListener(queues="queue01")
3  public class Consumer1 {
4      @RabbitHandler
5      public void process(String msg,Channel channel, Message message)
6      throws IOException {
7          try {
8              System.out.println("get msg1 success msg = "+msg);
9              /**
10               * 确认一条消息: <br>
11               * channel.basicAck(deliveryTag, false); <br>
12               * deliveryTag:该消息的index <br>
13               * multiple: 是否批量.true:将一次性ack所有小于deliveryTag的消息 <br>

```

```

13         */
14         channel.basicAck(message.getMessageProperties().getDeliveryTag(),
false);
15     } catch (Exception e) {
16         //消费者处理出了问题，需要告诉队列信息消费失败
17         /**
18          * 拒绝确认消息:<br>
19          * channel.basicNack(long deliveryTag, boolean multiple, boolean
requeue) ; <br>
20          * deliveryTag:该消息的index<br>
21          * multiple: 是否批量.true:将一次性拒绝所有小于deliveryTag的消息。<br>
22          * requeue: 被拒绝的是否重新入队列 <br>
23          */
24
25         channel.basicNack(message.getMessageProperties().getDeliveryTag(),
false, true);
26         System.err.println("get msg1 failed msg = "+msg);
27     }
28 }
29 }

```

10.5 消息消费的幂等性问题

消息消费的幂等性——多次消费的执行结果时相同的（避免重复消费）

解决方案：处理成功的消息setnx到redis

##

十一、延迟机制

11.1 延迟队列

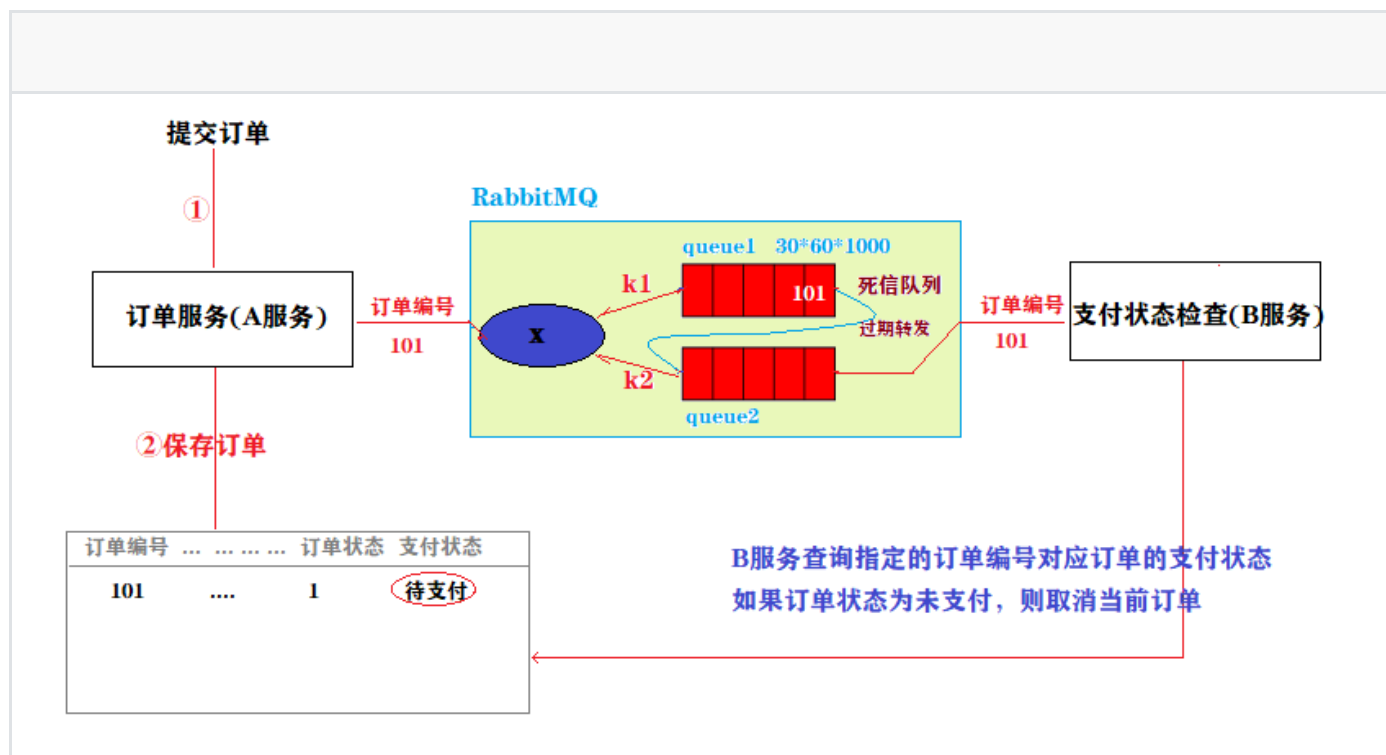
- 延迟队列——消息进入到队列之后，延迟指定的时间才能被消费者消费
- AMQP协议和RabbitMQ队列本身是不支持延迟队列功能的，但是可以通过TTL（Time To Live）特性模拟延迟队列的功能
- TTL就是消息的存活时间。RabbitMQ可以分别对队列和消息设置存活时间



- 在创建队列的时候可以设置队列的存活时间，当消息进入到队列并且在存活时间内没有消费者消费，则此消息就会从当前队列被移除；
- 创建消息队列没有设置TTL，但是消息设置了TTL，那么当消息的存活时间结束，也会被移除；
- 当TTL结束之后，我们可以指定将当前队列的消息转存到其他指定的队列

11.2 使用延迟队列实现订单支付监控

11.2.1 实现流程图



11.2.2 创建交换机和队列

1. 创建路由交换机

▼ Add a new exchange

Virtual host:

Name: *

Type:

Durability:

Auto delete: ?

Internal: ?

Arguments: =

Add **Alternate exchange** ?

Add exchange

2. 创建消息队列

▼ Add a new queue

Virtual host:

Name: *

Durability:

Auto delete: ?

Arguments: =

Add **Message TTL** ? | **Auto expire** ? | **Max length** ? | **Max length bytes** ? | **Overflow behaviour** ?
Dead letter exchange ? | **Dead letter routing key** ? | **Maximum priority** ?
Lazy mode ? **Master locator** ?

Add queue

3. 创建死信队列

▼ Add a new queue

Virtual host:

Name:

Durability:

Auto delete:

Arguments:

x-message-ttl	=	10000	设置死信队列存活时间	Number
x-dead-letter-exchange	=	delay_exchange	当消息过期之后转发到哪个交换机	String
x-dead-letter-routing-key	=	k2	当消息过期之后转发的交换机对应的key	String
	=			String

Add ☐ Message TTL ☐ Auto expire ☐ Max length ☐ Max length bytes ☐ Overflow behaviour ☐

☐ Dead letter exchange ☐ Dead letter routing key ☐ Maximum priority ☐

☐ Lazy mode ☐ Master locator ☐

Add queue

4. 队列绑定

This exchange



To	Routing key	Arguments	
delay_queue1	k1		Unbind
delay_queue2	k2		Unbind

十二、消息队列作用/使用场景总结

12.1 解耦

场景说明：用户下单之后，订单系统要通知库存系统

传统方式：订单系统直接调用库存系统提供的接口，如果库存系统出现故障会导致订单系统失败

订单系统

调用库存接口

库存系统

使用消息队列:



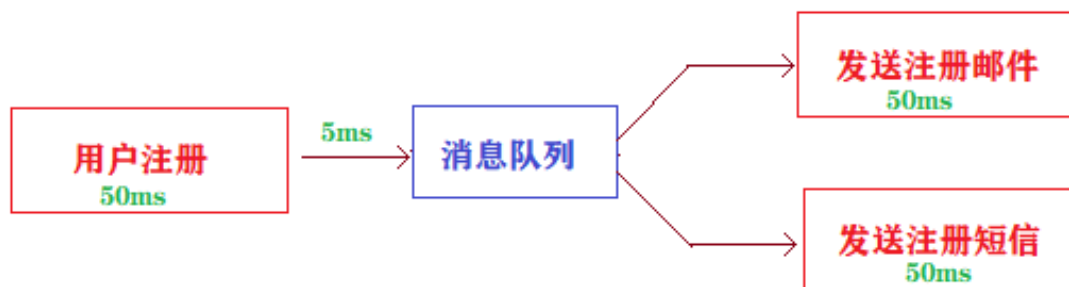
12.2 异步

场景说明: 用户注册成功之后, 需要发送注册邮件及注册短信提醒

传统方式:



使用消息队列:



12.3 消息通信

场景说明: 应用系统之间的通信, 例如聊天室

聊天室



12.4 流量削峰

场景说明：秒杀业务

大量的请求不会主动请求秒杀业务，而是存放在消息队列(缓存)



12.5 日志处理

场景说明：系统中大量的日志处理

日志搜集处理



千锋教育Java教研院 关注公众号【Java架构栈】 下载所有课程代码课件及工具 让技术回归本该有的纯静！