# RabbitMQ 1-工作模型与Java编程

## 课前准备

### 预习资料

Windows安装步骤

Linux安装步骤

官网文章中文翻译系列

### 环境说明

操作系统：CentOS 7

JDK：1.8

Erlang：19.0.4或最新版

RabbitMQ：3.6.12或最新版

版本对应关系

## 典型应用场景

1、跨系统的异步通信 人民银行二代支付系统，使用重量级消息队列 IBM MQ，异步，解耦，削峰都有体现。

2、应用内的同步变成异步 秒杀：自己发送给自己

3、基于Pub/Sub模型实现的事件驱动 放款失败通知、提货通知、购买碎屏保 系统间同步数据 摒弃ELT（比如全量同步商户数据）； 摒弃API（比如定时增量获取用户、获取产品，变成增量广播）。

4、利用RabbitMQ实现事务的最终一致性

## 基本介绍

### AMQP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/中间件同产品、不同的开发语言等条件的限制。

AMQP的实现有：RabbitMQ、OpenAMQ、Apache Qpid、Redhat Enterprise MRG、AMQP Infrastructure、ØMQ、Zyre等。

## RabbitMQ的特性

RabbitMQ使用Erlang语言编写，使用Mnesia数据库存储消息。

（1）可靠性（Reliability）RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。

（2）灵活的路由（Flexible Routing）在消息进入队列之前，通过 Exchange 来路由消息的。对于典型的路由功能，RabbitMQ 已经提供了一些内置的 Exchange 来实现。针对更复杂的路由功能，可以将多个 Exchange 绑定在一起，也通过插件机制实现自己的 Exchange 。

（3）消息集群（Clustering）多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker 。

（4）高可用（Highly Available Queues）队列可以在集群中的机器上进行镜像，使得在部分节点出问题的情况下队列仍然可用。

（5）多种协议（Multi-protocol）RabbitMQ 支持多种消息队列协议，比如 AMQP、STOMP、MQTT 等等。

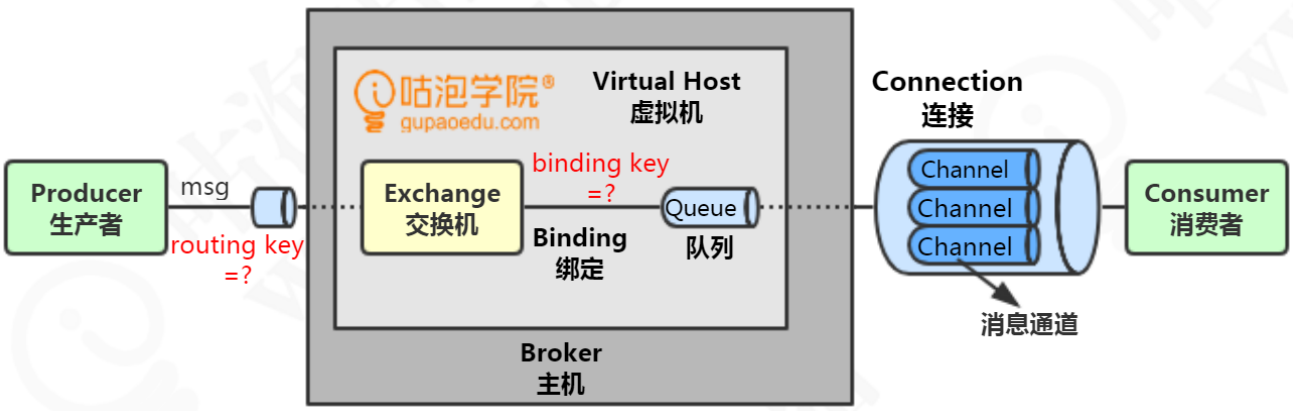
（6）多语言客户端（Many Clients）RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby、PHP、C#、JavaScript 等等。

（7）管理界面（Management UI）RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息、集群中的节点。

（8）插件机制（Plugin System）

RabbitMQ提供了许多插件，以实现从多方面扩展，当然也可以编写自己的插件。

## 工作模型



| 概念 | 解释 |
| --- | --- |
| Broker | 即RabbitMQ的实体服务器。提供一种传输服务，维护一条从生产者到消费者的传输线路，保证消息数据能按照指定的方式传输。 |
| Exchange | 消息交换机。指定消息按照什么规则路由到哪个队列Queue。 |
| Queue | 消息队列。消息的载体，每条消息都会被投送到一个或多个队列中。 |
| Binding | 绑定。作用就是将Exchange和Queue按照某种路由规则绑定起来。 |
| Routing Key | 路由关键字。Exchange根据Routing Key进行消息投递。定义绑定时指定的关键字称为Binding Key。 |
| Vhost | 虚拟主机。一个Broker可以有多个虚拟主机，用作不同用户的权限分离。一个虚拟主机持有一组Exchange、Queue和Binding。 |
| Producer | 消息生产者。主要将消息投递到对应的Exchange上面。一般是独立的程序。 |
| Consumer | 消息消费者。消息的接收者，一般是独立的程序。 |
| Connection | Producer 和 Consumer 与Broker之间的TCP长连接。 |
| Channel | 消息通道，也称信道。在客户端的每个连接里可以建立多个Channel，每个Channel代表一个会话任务。在RabbitMQ Java Client API中，channel上定义了大量的编程接口。 |

# 三种主要的交换机

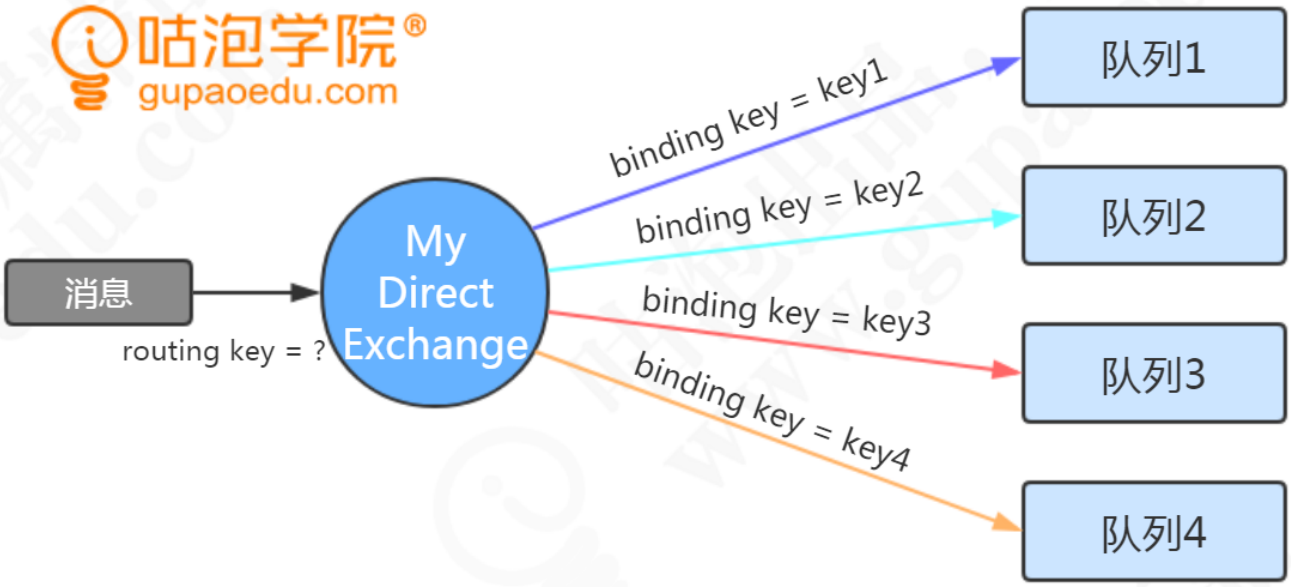
## Direct Exchange 直连交换机

定义：直连类型的交换机与一个队列绑定时，需要指定一个明确的binding key。

路由规则：发送消息到直连类型的交换机时，只有routing key跟binding key完全匹配时，绑定的队列才能收到消息。

例如：

```
//  只有队列1能收到消息
channel.basicPublish("MY_DIRECT_EXCHANGE", "key1", null, msg.getBytes());
```




### Topic Exchange 主题交换机

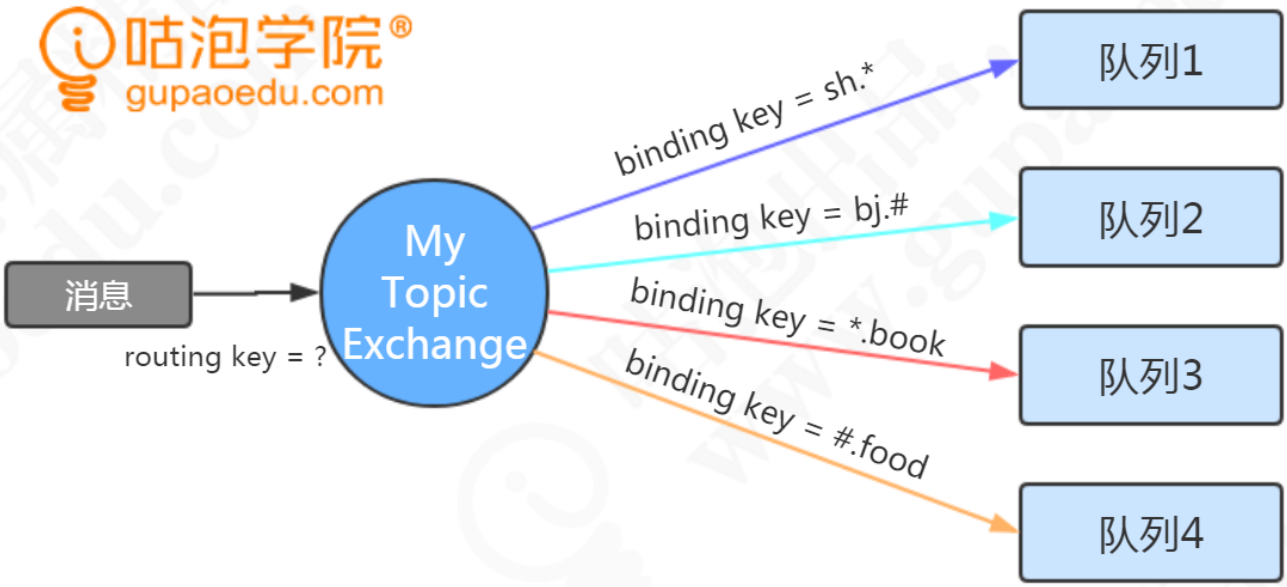
定义：主题类型的交换机与一个队列绑定时，可以指定按模式匹配的routing key。

通配符有两个，*代表匹配一个单词。#代表匹配零个或者多个单词。单词与单词之间用 . 隔开。

路由规则：发送消息到主题类型的交换机时，routing key符合binding key的模式时，绑定的队列才能收到消息。

例如：

```
// 只有队列1能收到消息
channel.basicPublish("MY_TOPIC_EXCHANGE", "sh.abc", null, msg.getBytes());
// 队列2和队列3能收到消息
channel.basicPublish("MY_TOPIC_EXCHANGE", "bj.book", null, msg.getBytes());
// 只有队列4能收到消息
channel.basicPublish("MY_TOPIC_EXCHANGE", "abc.def.food", null, msg.getBytes());
```



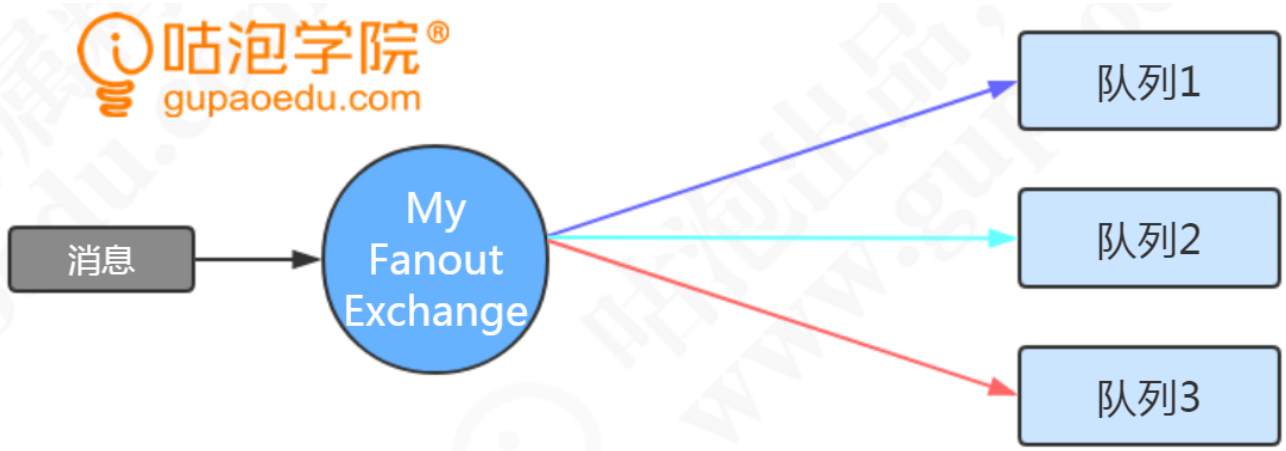

### Fanout Exchange 广播交换机

定义：广播类型的交换机与一个队列绑定时，不需要指定binding key。

路由规则：当消息发送到广播类型的交换机时，不需要指定routing key，所有与之绑定的队列都能收到消息。

例如：

```
// 3个队列都会收到消息
channel.basicPublish("MY_FANOUT_EXCHANGE", "", null, msg.getBytes());
```




## Java API 编程

### 创建Maven工程，pom.xml引入依赖

```
        <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
            <version>4.1.0</version>
        </dependency>
```

## 生产者

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

public class MyProducer {
    private final static String QUEUE_NAME = "ORIGIN_QUEUE";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        // 连接IP
        factory.setHost("127.0.0.1");
        // 连接端口
        factory.setPort(5672);
        // 虚拟机
        factory.setVirtualHost("/");
        // 用户
        factory.setUsername("guest");
        factory.setPassword("guest");

        // 建立连接
        Connection conn = factory.newConnection();
        // 创建消息通道
        Channel channel = conn.createChannel();

        String msg = "Hello world, Rabbit MQ";
        // 声明队列
        // String queue, boolean durable, boolean exclusive, boolean autoDelete,
Map<String, Object> arguments
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 发送消息（发送到默认交换机AMQP Default，Direct）
        // 如果有一个队列名称跟Routing Key相等，那么消息会路由到这个队列
        // String exchange, String routingKey, BasicProperties props, byte[] body
        channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());

        channel.close();
        conn.close();
    }
}
```

## 消费者

```
import com.rabbitmq.client.*;
import java.io.IOException;

public class MyConsumer {
    private final static String QUEUE_NAME = "ORIGIN_QUEUE";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        // 连接IP
        factory.setHost("127.0.0.1");
        // 默认监听端口
        factory.setPort(5672);
        // 虚拟机
        factory.setVirtualHost("/");
        // 设置访问的用户
        factory.setUsername("guest");
        factory.setPassword("guest");

        // 建立连接
        Connection conn = factory.newConnection();
        // 创建消息通道
        Channel channel = conn.createChannel();

        // 声明队列
        // String queue, boolean durable, boolean exclusive, boolean autoDelete,
Map<String, Object> arguments
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        System.out.println(" Waiting for message....");

        // 创建消费者
        Consumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                String msg = new String(body, "UTF-8");
                System.out.println("Received message : '" + msg + "'");
            }
        };

        // 开始获取消息
        // String queue, boolean autoAck, Consumer callback
        channel.basicConsume(QUEUE_NAME, true, consumer);
    }
}
```

## 参数说明

### 声明交换机的参数

String type：交换机的类型，direct, topic, fanout中的一种。

boolean durable：是否持久化，代表交换机在服务器重启后是否还存在。

**声明队列的参数**

boolean durable：是否持久化，代表队列在服务器重启后是否还存在。

boolean exclusive：是否排他性队列。排他性队列只能在声明它的Connection中使用，连接断开时自动删除。

boolean autoDelete：是否自动删除。如果为true，至少有一个消费者连接到这个队列，之后所有与这个队列连接的消费者都断开时，队列会自动删除。

Map<String, Object> arguments：队列的其他属性，例如x-message-ttl、x-expires、x-max-length、x-max-length-bytes、x-dead-letter-exchange、x-dead-letter-routing-key、x-max-priority。

**消息属性BasicProperties**

消息的全部属性有14个，以下列举了一些主要的参数：

| 参数 | 释义 |
| --- | --- |
| Map<String,Object> headers | 消息的其他自定义参数 |
| Integer deliveryMode | 2持久化，其他：瞬态 |
| Integer priority | 消息的优先级 |
| String correlationId | 关联ID，方便RPC相应与请求关联 |
| String replyTo | 回调队列 |
| String expiration | TTL，消息过期时间，单位毫秒 |

# 进阶知识

## 1、TTL（Time To Live）

### a、消息的过期时间

有两种设置方式：

通过队列属性设置消息过期时间：

```
Map<String, Object> argss = new HashMap<String, Object>();
argss.put("x-message-ttl",6000);

channel.queueDeclare("TEST_TTL_QUEUE", false, false, false, argss);
```

设置单条消息的过期时间：

```
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
        .deliveryMode(2) // 持久化消息
        .contentEncoding("UTF-8")
        .expiration("10000") // TTL
        .build();

channel.basicPublish("", "TEST_TTL_QUEUE", properties, msg.getBytes());
```

### b、队列的过期时间

```
Map<String, Object> argss = new HashMap<String, Object>();
argss.put("x-message-ttl",6000);

channel.queueDeclare("TEST_TTL_QUEUE", false, false, false, argss);
```

队列的过期时间决定了在没有任何消费者以后，队列可以存活多久。

参考：

com.gupaoedu.ttl

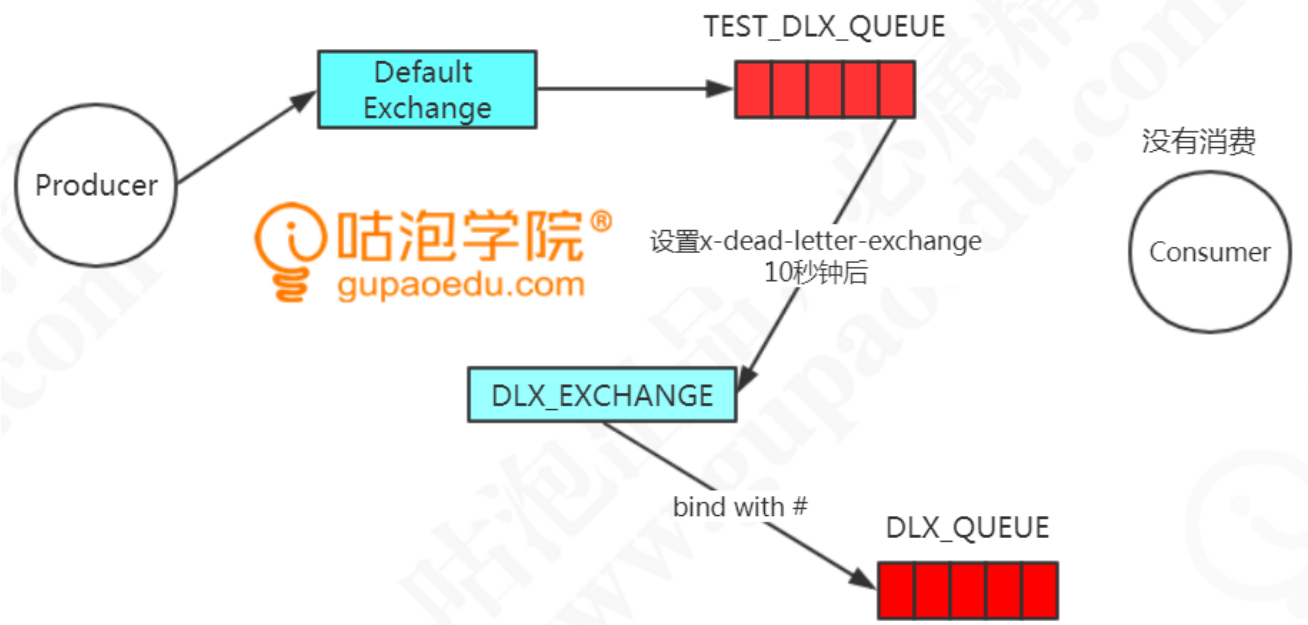
## 2、死信队列

有三种情况消息会进入DLX（Dead Letter Exchange）死信交换机。

1、(NACK || Reject ) && requeue == false

2、消息过期

3、队列达到最大长度（先入队的消息会被发送到DLX）




可以设置一个死信队列（Dead Letter Queue）与DLX绑定，即可以存储Dead Letter，消费者可以监听这个队列取走消息。

```
Map<String,Object> arguments = new HashMap<String,Object>();
arguments.put("x-dead-letter-exchange","DLX_EXCHANGE");
// 指定了这个队列的死信交换机
channel.queueDeclare("TEST_DLX_QUEUE", false, false, false, arguments);

// 声明死信交换机
channel.exchangeDeclare("DLX_EXCHANGE","topic", false, false, false, null);
// 声明死信队列
channel.queueDeclare("DLX_QUEUE", false, false, false, null);
// 绑定
channel.queueBind("DLX_QUEUE","DLX_EXCHANGE","#");
```

参考：

com.gupaoedu.dlx

## 3、优先级队列

设置一个队列的最大优先级：

```
Map<String, Object> argss = new HashMap<String, Object>();
argss.put("x-max-priority",10);  // 队列最大优先级

channel.queueDeclare("ORIGIN_QUEUE", false, false, false, argss);
```

发送消息时指定消息当前的优先级：

```
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
        .priority(5)  // 消息优先级
        .build();

channel.basicPublish("", "ORIGIN_QUEUE", properties, msg.getBytes());
```

优先级高的消息可以优先被消费，但是：只有消息堆积（消息的发送速度大于消费者的消费速度）的情况下优先级才有意义。

参考：

com.gupaoedu.message

## 4、延迟队列

RabbitMQ本身不支持延迟队列。可以使用TTL结合DLX的方式来实现消息的延迟投递，即把DLX跟某个队列绑定，到了指定时间，消息过期后，就会从DLX路由到这个队列，消费者可以从这个队列取走消息。

另一种方式是使用rabbitmq-delayed-message-exchange插件。

当然，将需要发送的信息保存在数据库，使用任务调度系统扫描然后发送也是可以实现的。

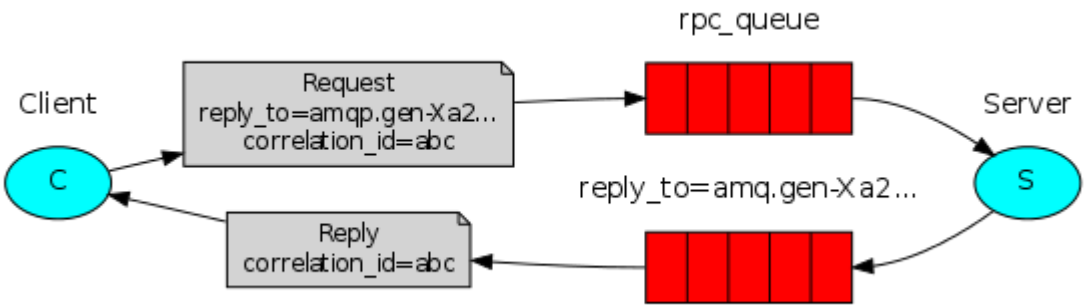
参考：

com.gupaoedu.dlx

## 5、RPC

RabbitMQ实现RPC的原理：服务端处理消息后，把响应消息发送到一个响应队列，客户端再从响应队列取到结果。

其中的问题：Client收到消息后，怎么知道应答消息是回复哪一条消息的？所以必须有一个唯一ID来关联，就是correlationId。



参考：

com.gupaoedu.rpc

## 6、服务端流控（Flow Control）

RabbitMQ 会在启动时检测机器的物理内存数值。默认当 MQ 占用 40% 以上内存时，MQ 会主动抛出一个内存警告并阻塞所有连接（Connections）。可以通过修改 rabbitmq.config 文件来调整内存阈值，默认值是 0.4，如下所示： [{rabbit, [{vm_memory_high_watermark, 0.4}]}].

默认情况，如果剩余磁盘空间在 1GB 以下，RabbitMQ 主动阻塞所有的生产者。这个阈值也是可调的。

注意队列长度只在消息堆积的情况下有意义，而且会删除先入队的消息，不能实现服务端限流。

## 7、消费端限流

在AutoACK为false的情况下，如果一定数目的消息（通过基于consumer或者channel设置Qos的值）未被确认前，不进行消费新的消息。

```
channel.basicQos(2); // 如果超过2条消息没有发送ACK，当前消费者不再接受队列消息
channel.basicConsume(QUEUE_NAME, false, consumer);
```

参考：com.gupaoedu.limit

# UI管理界面的使用

管理插件提供了更简单的管理方式。

### 启用管理插件

#### Windows启用管理插件

```
cd C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.6\sbin
rabbitmq-plugins.bat enable rabbitmq_management
```

#### Linux启用管理插件

```
cd /usr/lib/rabbitmq/bin
./rabbitmq-plugins enable rabbitmq_management
```

### 管理界面访问端口

默认端口是15672，默认用户guest，密码guest。guest用户默认只能在本机访问。

### Linux 创建RabbitMQ用户

例如创建用户admin，密码admin，授权访问所有的Vhost

```
firewall-cmd --permanent --add-port=15672/tcp
firewall-cmd --reload
rabbitmqctl add_user admin admin
rabbitmqctl set_user_tags admin administrator
rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"
```

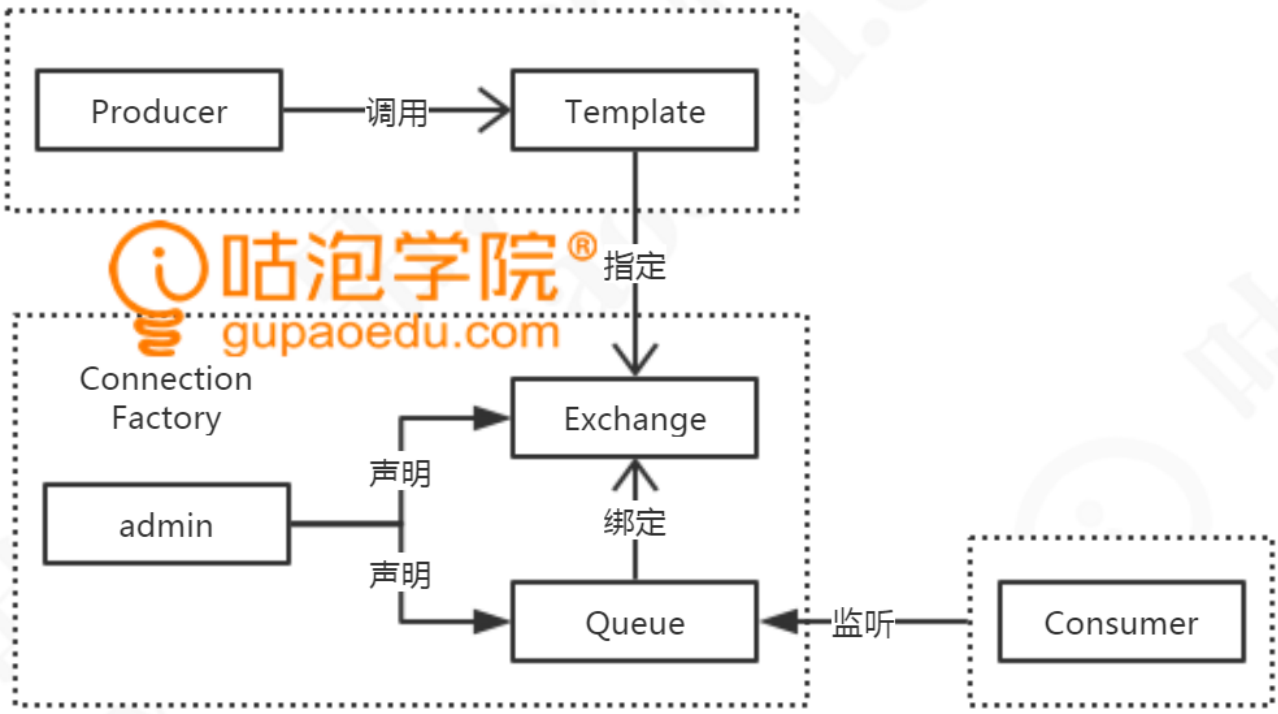
## Spring配置方式集成RabbitMQ

参考gitlab工程

步骤：

1、创建Maven工程，pom.xml引入依赖

2、src/main/resouces目录，创建rabbitMQ.xml




3、配置applicationContext.xml

4、src/main/resouces目录，log4j.properties

5、编写生产者

6、编写4个消费者

7、编写单元测试类

# Spring Boot集成RabbitMQ

参考gitlab工程