



消息中间件的分类：

协议 AMQP 。

1：PUSH 推模型 消息生产者将消息发送给消息传递服务，消息服务又将消息推送给消息的消费者。一般实时性比较高。

2：PULL 拉消息模型 消费者请求消息服务接受消息，

实时性可能不高。消费者从消息中间件中拉取消息。

RabbitMQde 消息存储在队列中，交换机的使用并不真正消耗服务器的性能，而队列会。衡量RabbitMq的QPS(每秒查询率，规定时间内锁处理流量多少的衡量标准)只需看队列即可。

生产中药对创建的队列的流量，内存占用和网卡有一个清晰的认识，预估平均值和峰值，以便合理分配硬件资源。

具体区别：



推和拉的使用场景：

推：

缓存数据同步更新

往应用推数据

发送消息：

Void basicPublish(){

}



拉：

1：把消息中间件当成可靠的消息暂存池。

2：定时进行消息投递，比如用户秒杀访问，进行系统性能压力测试。



核心，看时间，看实时性的要求。

mandatory参数：

当mandatory参数为true的时候，如果交换机无法根据自身的类型和路由键找到一个符合条件的队列，那么RabbitMq会调动Basic.Return命令将消息返回给生产者。当参数mandatory为false时，出现上述情形，则消息直接丢弃。

immediate属性在Rabbitmq3.x的时候，被废弃了。

当immediate参数为true的时候，如果交换器在将消息路由到队列时发现队列上并不存在任何消费者，  
因为这个关键字违背了生产者和消费者之间解耦的特性，因为生产者不关心消息是否被消费者消费掉，所以这个字段被drop掉了。官方的解释是 immediate参数会影响镜像队列的性能，增加了代码的复杂性，建议采用TTL和DLX的方法替代。

**生产者：**

public class Prvoider {

private static final String QUEUENAME = "q1";

private static final String EXECHANGENAME = "c1";

private static final String ROUTENAME = "r1";

public static void main(String[] args) throws Exception {

Connection conn = ConnectionUtil.getConnection();

// 创建消息的频道

Channel chanel = conn.createChannel();

// 声明一个队列

// chanel.exchangeDeclare(EXECHANGENAME, "direct");

// Map map = new HashMap<String,Object>();

// map.put("x-max-length", 2);

// map.put("x-overflow", "reject-publish");

// chanel.queueDeclare(QUEUENAME, false, false, false, null);

chanel.addReturnListener(new ReturnListener() {

@Override

public void handleReturn(int arg0, String arg1, String arg2, String arg3, BasicProperties arg4, byte[] body)

throws IOException {

// TODO Auto-generated method stub

System.out.println(arg0);

String msg = new String(body);

System.out.println("Basic.Return返回的结果是 " + msg);

}

});

//

for (int i = 0; i < 5; i++) {

String msg = "你没吃---饭吗？ ";

chanel.basicPublish(EXECHANGENAME, "r2", true, MessageProperties.TEXT\_PLAIN, msg.getBytes());

System.out.println("消息发布成功");

}

Thread.sleep(2000);

conn.close();

}

}

**消费者：**

public class Customer {

private static final String QUEUENAME = "q1";

private static final String EXECHANGENAME = "c1";

private static final String ROUTENAME = "r1";

public static void main(String[] args) throws Exception {

Connection conn = ConnectionUtil.getConnection();

//创建消息的频道

Channel chanel = conn.createChannel();

chanel.exchangeDeclare(EXECHANGENAME, "topic");

//chanel.exchangeDeclare(EXECHANGENAME, "topic", true, false, null);

//声明一个队列

// Map map = new HashMap<String,Object>();

// map.put("x-max-length", 20);

// map.put("x-overflow", "reject-publish");

chanel.queueDeclare(QUEUENAME, false,false,false, null);

chanel.queueBind(QUEUENAME, EXECHANGENAME, ROUTENAME);

chanel.basicQos(1);//限流 每次只从服务器上取1条消息

DefaultConsumer consumer = new DefaultConsumer(chanel) {

@Override

public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties, byte[] body)

throws IOException {

// TODO Auto-generated method stub

String msg = new String(body);

try {

Thread.sleep(1);

chanel.basicAck(envelope.getDeliveryTag(), false);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("接受到的消息"+msg);

}

};

boolean autoAck = false;//设置消息的自动确认

chanel.basicConsume(QUEUENAME, false, consumer);

}

}

备份交换器：

生产者在发送消息的时候如果不设置mandatory参数，那么消息在未被路由的情况下将被丢失，如果设置了mandatory参数，又需要添加ReturnListener的代码，生产者代码结构变得复杂。

如果又不想要复杂的编程逻辑，又不想消息丢失，可以使用备份交换器。将未被路由的消息存储在RabbitMq中，需要的时候再来处理。