# rabbitMQ

## 基础

message queen

what is rabbitmq?

通过生产者消费者的模型去收发消息内容

#1.ActiveMQ

性能和容量规模相对不足

#2.Kafka

追求高吞吐量，不支持事务，数据重复丢失无限制

#3.RocketMQ

多功能收费

#4.RabbitMQ

why

解决各服务之间的耦合问题，高吞吐量接收和发送

## 消息模型

### hellowolrd

1. */\*\*  
    \* 生产者  
    \*/*public class Send {  
     
    private final static String *QUEUE\_NAME* = "simple\_queue";  
     
    public static void main(String[] argv) throws Exception {  
    // 获取到连接  
    Connection connection = ConnectionUtil.*getConnection*();  
    // 从连接中创建通道，使用通道才能完成消息相关的操作  
    Channel channel = connection.createChannel();  
    // 声明（创建）队列  
    channel.queueDeclare(*QUEUE\_NAME*, false, false, false, null);  
    // 消息内容  
    String message = "Hello World!";  
    // 向指定的队列中发送消息  
    channel.basicPublish("", *QUEUE\_NAME*, null, message.getBytes());  
      
    System.*out*.println(" [x] Sent '" + message + "'");  
     
    //关闭通道和连接  
    channel.close();  
    connection.close();  
    }  
   }

自动ACK消费消息

*/\*\*  
 \* 消费者  
 \*/*public class Recv {  
 private final static String *QUEUE\_NAME* = "simple\_queue";  
  
 public static void main(String[] argv) throws Exception {  
 // 获取到连接  
 Connection connection = ConnectionUtil.*getConnection*();  
 // 创建通道  
 Channel channel = connection.createChannel();  
 // 声明队列 重复声明不会报错 名字一致属性也必须一致  
 channel.queueDeclare(*QUEUE\_NAME*, false, false, false, null);  
 // 定义队列的消费者  
 DefaultConsumer consumer = new DefaultConsumer(channel) {  
 // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用  
 @Override  
 public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,  
 byte[] body) throws IOException {  
 // body 即消息体  
 String msg = new String(body);  
 System.*out*.println(" [x] received : " + msg + "!");  
 }  
 };  
 // 监听队列，第二个参数：是否自动进行消息确认。  
 channel.basicConsume(*QUEUE\_NAME*, true, consumer);  
 }  
}

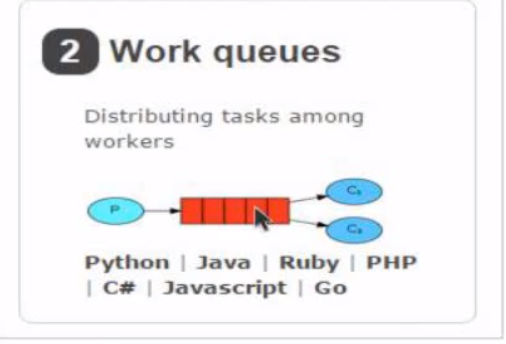
防止消息丢失的一种方案

手动ACK确认消费（防止消费者中断，mq又丢失了该条消失）

*/\*\*  
 \* 消费者,手动进行ACK  
 \*/*public class Recv2 {  
 private final static String *QUEUE\_NAME* = "simple\_queue";  
  
 public static void main(String[] argv) throws Exception {  
 // 获取到连接  
 Connection connection = ConnectionUtil.*getConnection*();  
 // 创建通道  
 final Channel channel = connection.createChannel();  
 // 声明队列  
 channel.queueDeclare(*QUEUE\_NAME*, false, false, false, null);  
 // 定义队列的消费者  
 DefaultConsumer consumer = new DefaultConsumer(channel) {  
 // 获取消息，并且处理，这个方法类似事件监听，如果有消息的时候，会被自动调用  
 @Override  
 public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties,  
 byte[] body) throws IOException {  
 // body 即消息体  
 String msg = new String(body);  
 System.*out*.println(" [x] received : " + msg + "!");  
 // 手动进行ACK  
 channel.basicAck(envelope.getDeliveryTag(), false);  
 }  
 };  
 // 监听队列，第二个参数false，手动进行ACK  
 channel.basicConsume(*QUEUE\_NAME*, false, consumer);  
 }  
}

### work消息模型

消费者不足导致消息堆积，增加消费者，一个队列绑定多个消费者

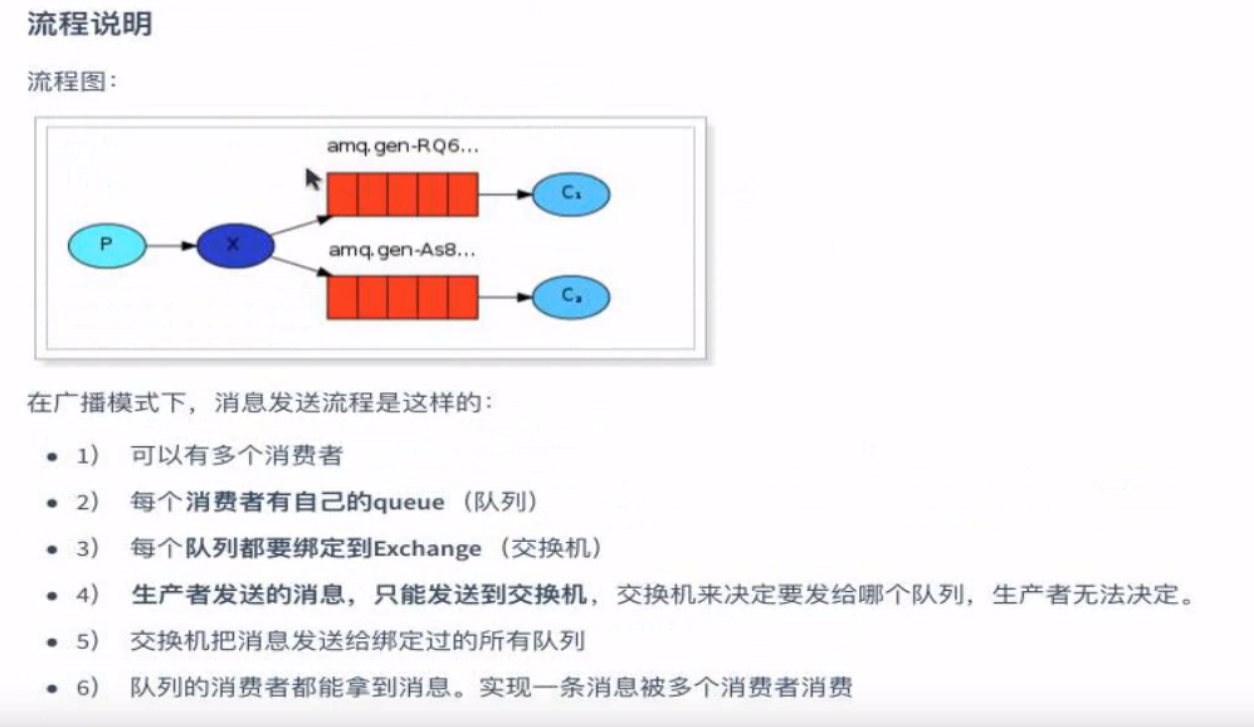


### 发布订阅模型

#### fanout

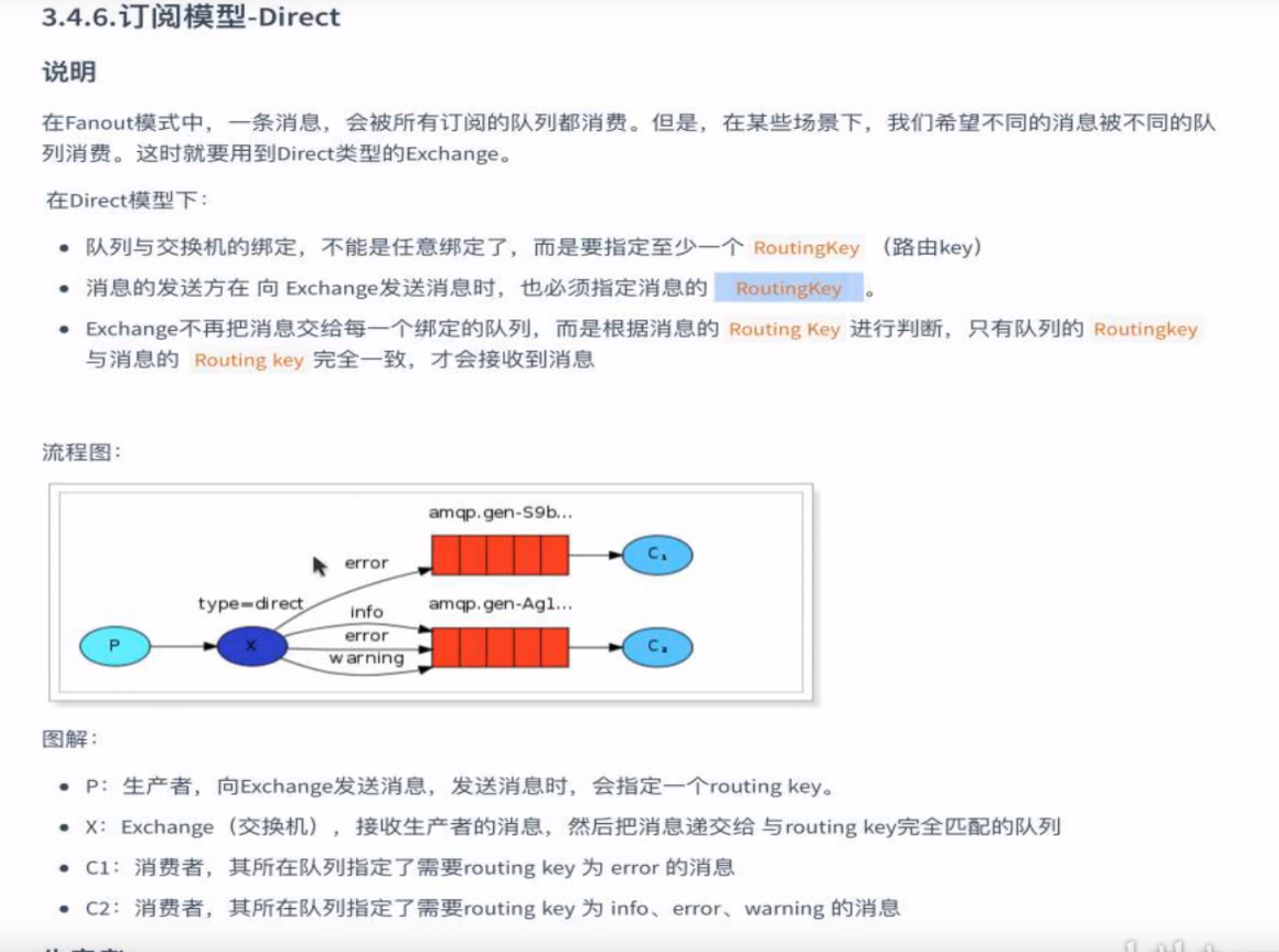
广播

生产者 → 交换器 → 队列+ → 消费者



#### Direct

与fanout区别在于根据生产者方发送不同消息（）routingKey）发送到不同队列，然后再给到消费者

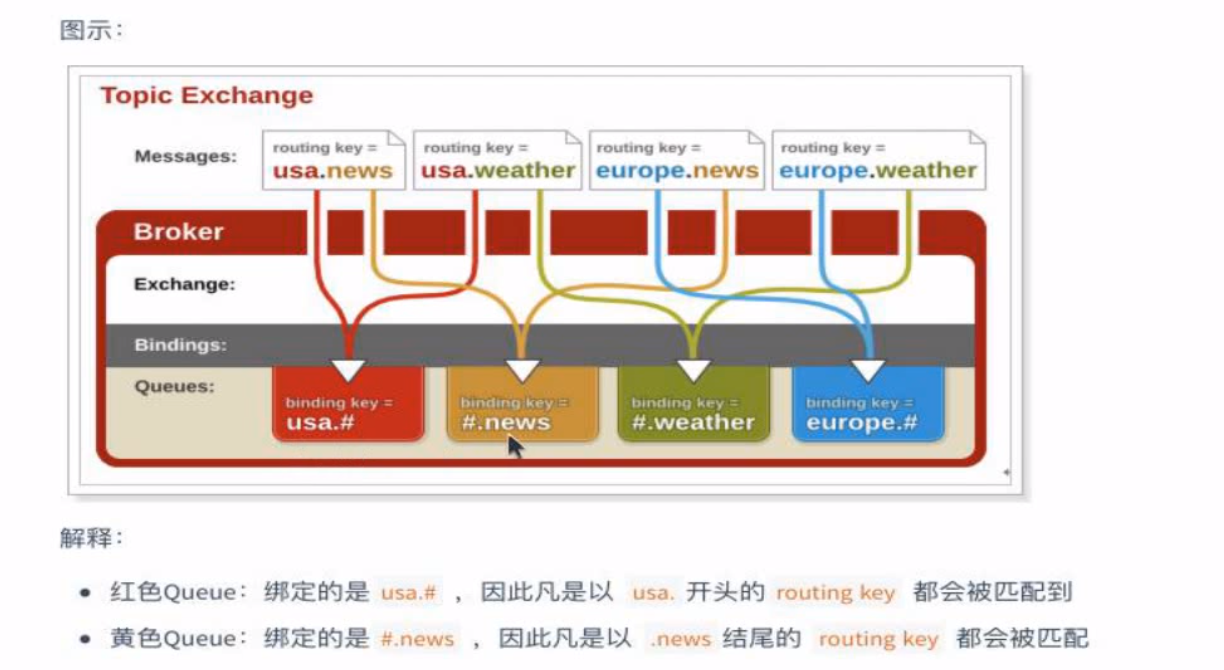


#### Topic

与Direct区别

通配符规则





### 与spring整合AMQP

application.yml配置

**spring:  
 rabbitmq:  
 host:** 192.168.1.206  
 **username:** leyou  
 **password:** leyou  
 **virtual-host:** /leyou

启动类（无关）

@SpringBootApplication  
public class Application {  
 public static void main(String[] args) { SpringApplication.*run*(Application.class, args); }  
}

发送消息

@RunWith(SpringRunner.class)  
@SpringBootTest(classes = Application.class)  
public class MqDemo {  
 @Autowired  
 private AmqpTemplate amqpTemplate;  
 @Test  
 public void testSend() throws InterruptedException {  
 String msg = "hello, Spring boot amqp";  
 this.amqpTemplate.convertAndSend("spring.test.exchange","a.b", msg);  
 // 等待10秒后再结束  
 Thread.*sleep*(10000);  
 }  
}

接收消息

@Component  
public class Listener {  
 @RabbitListener(bindings = @QueueBinding(  
 value = @Queue(value = "spring.test.queue", durable = "true"),  
 exchange = @Exchange(  
 value = "spring.test.exchange",  
 ignoreDeclarationExceptions = "true",  
 type = ExchangeTypes.*TOPIC* ),  
 key = {"#.#"}))  
 public void listen(String msg){  
 System.*out*.println("接收到消息：" + msg);  
 }  
}

### 案例

防止消息丢失

1. ACK消费者确认，业务成功处理才进行提交，mq才将消息删除（防止消费者弄丢）
2. 持久化（防止MQ弄丢）



1. 生产者确认（防止生产者弄丢）
2. 发送消息前将消息存到数据库，保证可靠性

幂等性：（统一接口被重复执行，其结果一致）