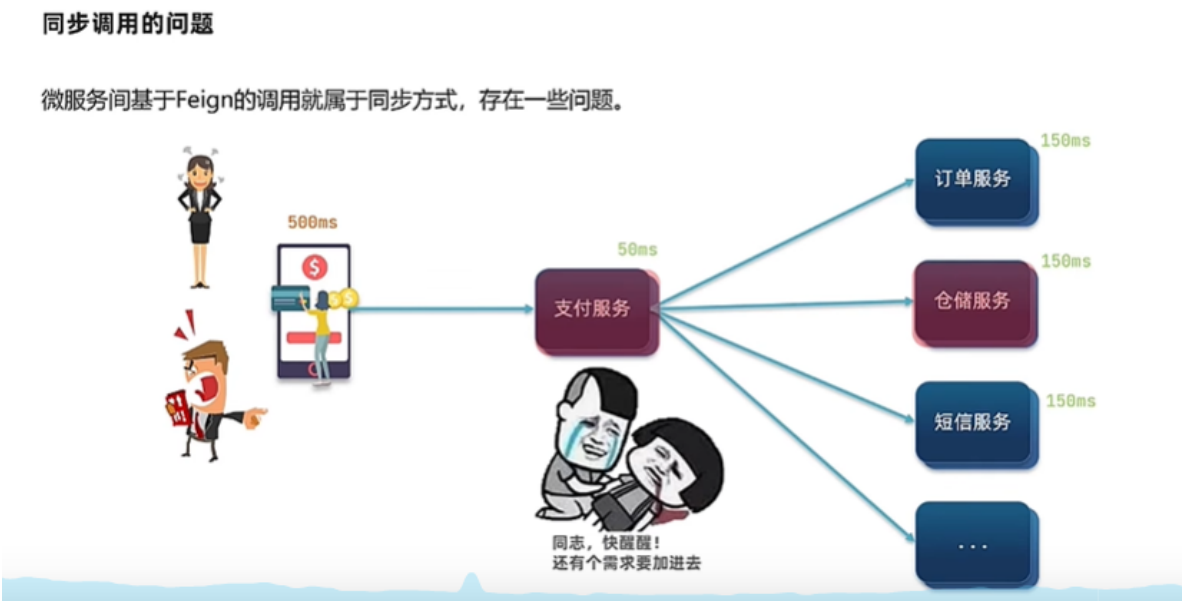
RabbitMq

1 同步通讯优缺点



优点：时效性高

缺点：

1 耦合度高，每次加入新需求都需要改动代码

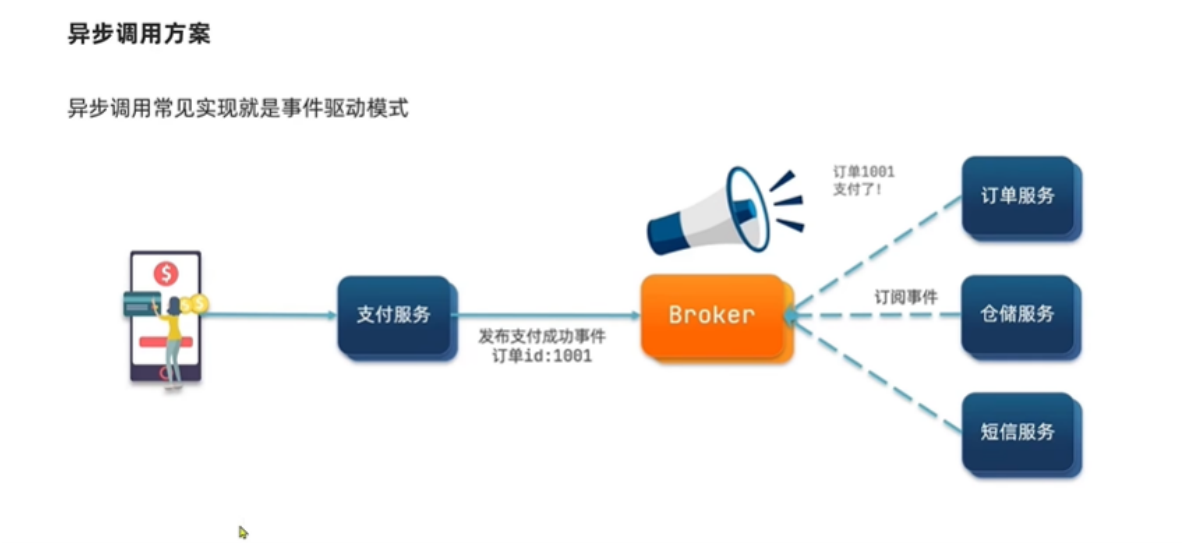
2 性能低，吞吐量低

3 一次只能调用一个服务，占着cpu资源，资源浪费率高，

4 级联实现，某个服务挂掉后容易引起阻塞（级联失败）

2 异步通讯优缺点

常见实现方式：事件驱动模式（由事件通知其它服务）



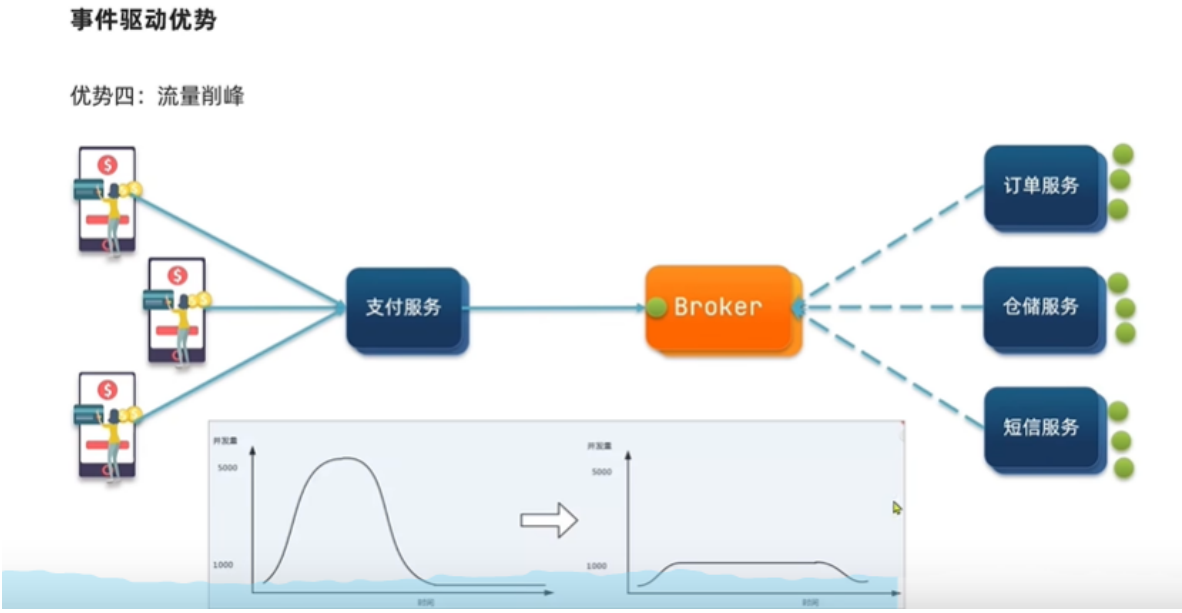
***优点**

1 服务解耦

2 性能提升，吞吐量提高

3 服务没有强依赖，没有级联失败的问题

4 流量消峰（由事件来进行控制）



缺点

1 依赖于Broker的可靠应、安全性、吞吐能力

2 架构复杂了、业务没有明显的流程性不好追踪管理

3 什么是MQ

MQ(MessageQueue)就是消息队列，即存放消息的队列，也就是事件驱动架构中的Broker

4 常用的MQ及区别

MQ （MessageQueue），中文是消息队列，字面来看就是存放消息的队列。也就是事件驱动架构中的Broker。

	RabbitMQ	ActiveMQ	RocketMQ	Kafka
公司/社区	Rabbit	Apache	阿里	Apache
开发语言	Erlang	Java	Java	Scala&Java
协议支持	AMQP，XMPP，SMTP，STOMP	OpenWire,STOMP，REST,XMPP,AMQP	自定义协议	自定义协议
可用性	高	一般	高	高
单机吞吐量	一般	差	高	非常高
消息延迟	微秒级	毫秒级	毫秒级	毫秒以内
消息可靠性	高	一般	高	一般

5 RabbitMq几个概念

RabbitMQ中的几个概念：

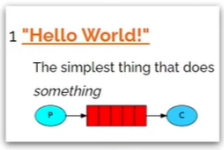
- channel：操作MQ的工具
- exchange：路由消息到队列中
- queue：缓存消息
- virtual host：虚拟主机，是对queue、exchange等资源的逻辑分组

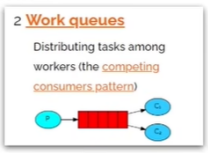
6 消息队列模型

常见消息模型

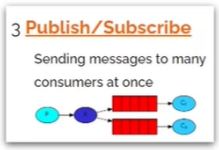
MQ的官方文档中给出了5个MQ的Demo示例，对应了几种不同的用法：

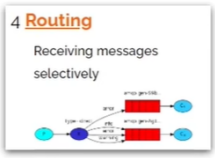
- 基本消息队列（BasicQueue）
- 工作消息队列（WorkQueue）

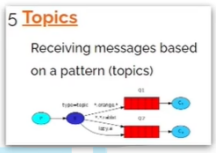




- 发布订阅（Publish、Subscribe），又根据交换机类型不同分为三种：
  - Fanout Exchange：广播
  - Direct Exchange：路由
  - Topic Exchange：主题







1 简单队列模型

生产者

1 创建连接，设置参数，建立连接

```
// 1.建立连接
ConnectionFactory factory = new ConnectionFactory();  factory: ConnectionFactory@1831
// 1.1.设置连接参数，分别是：主机名、端口号、vhost、用户名、密码
factory.setHost("192.168.150.101");
factory.setPort(5672);
factory.setVirtualHost("/");
factory.setUsername("itcast");
factory.setPassword("123321");
// 1.2.建立连接
Connection connection = factory.newConnection();  factory: ConnectionFactory@1831
```

2 创建通道，创建队列

```
// 2.创建通道Channel
Channel channel = connection.createChannel();  connection: "amqp://itcast@192.168.150.101:5672/"

// 3.创建队列
String queueName = "simple.queue";
channel.queueDeclare(queueName, durable: false, exclusive: false, autoDelete: false, arguments: null);
```

3 发送消息并关闭连接

```
// 4.发送消息
String message = "hello, rabbitmq!";
channel.basicPublish( exchange: "", queueName, props: null, message.getBytes());
System.out.println("发送消息成功：【" + message + "】");

// 5.关闭通道和连接
channel.close();
connection.close();
```

消费者

1 创建连接，设置参数，建立连接

2 创建通道，创建队列

3 订阅消息

```
// 4.订阅消息
channel.basicConsume(queueName, autoAck: true, new DefaultConsumer(channel){  channel: "AMQChannel(amq
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
                               AMQP.BasicProperties properties, byte[] body) throws IOException {
        // 5.处理消息
        String message = new String(body);
        System.out.println("接收到消息：【" + message + "】");
    }
});
System.out.println("等待接收消息。。。。");
}
```

7 什么是Spring AMQP

AMQP是消息队列的一种规范协议。

Spring AMQP是基于AMQP协议定义的一套Api规范，提供了模板来发送和接受消息。(类似于Redis的RedisTemplate)

包含两部分：spring-amqp是基础抽象 spring-rabbit是底层实现

8 Spring AMQP实现简单队列

基本流程

流程如下：

1. 在父工程中引入spring-amqp的依赖
2. 在publisher服务中利用RabbitTemplate发送消息到simple.queue这个队列
3. 在consumer服务中编写消费逻辑，绑定simple.queue这个队列

依赖

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

生产者yml文件及测试类

1. 在publisher服务中编写application.yml，添加mq连接信息：

```
spring:
  rabbitmq:
    host: 192.168.150.101 # 主机名
    port: 5672 # 端口
    virtual-host: / # 虚拟主机
    username: itcast # 用户名
    password: 123321 # 密码
```

2. 在publisher服务中新建一个测试类，编写测试方法：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringAmqpTest {
    @Autowired
    private RabbitTemplate rabbitTemplate;
    @Test
    public void testSimpleQueue() {
        String queueName = "simple.queue";
        String message = "hello, spring amqp!";
        rabbitTemplate.convertAndSend(queueName, message);
    }
}
```

注意：此方法不会自动创建队列，队列必须已经存在

SpringAMQP接收消息

1 依赖，yml文件。与生产者类似

2 创建消费逻辑，监听队列

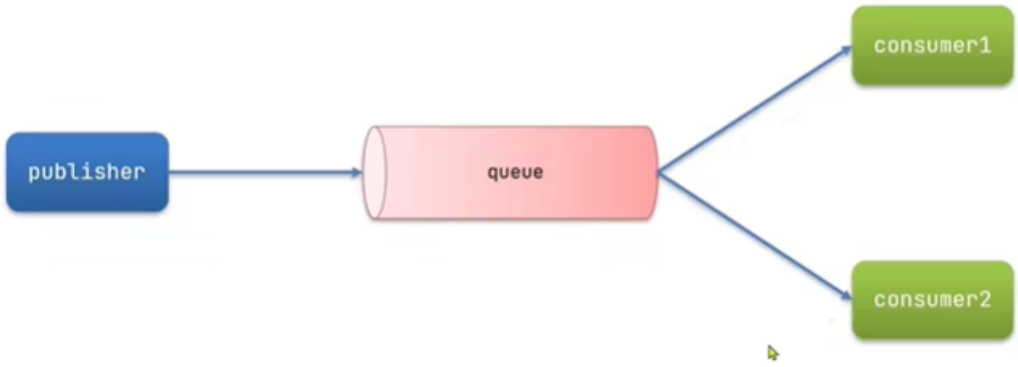
2. 在consumer服务中新建一个类，编写消费逻辑：

```
@Component
public class SpringRabbitListener {

    @RabbitListener(queues = "simple.queue")
    public void listenSimpleQueueMessage(String msg) throws InterruptedException {
        System.out.println("spring 消费者接收到消息 : [" + msg + "] ");
    }
}
```

注意：消息一旦消费就会从队列中删除，RabbitMq没有消息回溯功能

9 Spring AMQP实现工作队列

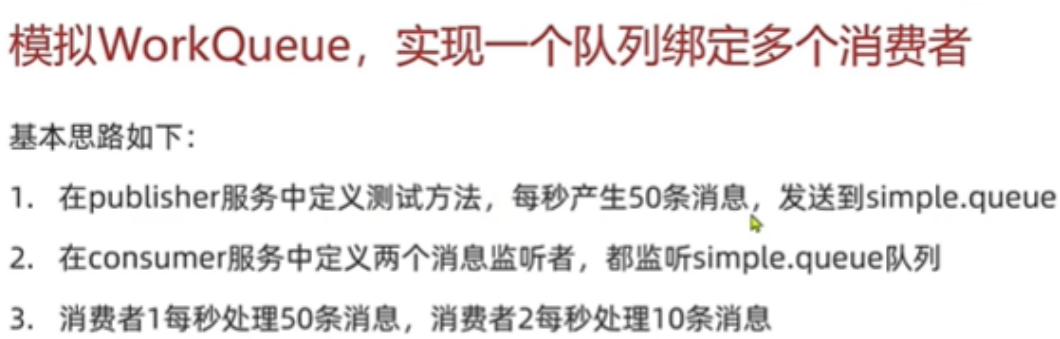


一个队列后跟着两个消费者

优点：提高消息处理速度，避免队列消息堆积

注意：同一条消息只会被一个消费者消费

模拟工作队列



生产者

```
//模拟工作队列
    @Test
    public void workQueueAMQP() throws InterruptedException {
        String queueName="simple.queue";
        String messqge = "WorkQueue SpringAMQP __";
        for (int i=1;i<=50;i++){
            rabbitTemplate.convertAndSend(queueName,messqge+i);
            Thread.sleep(20);
        }

    }
```

消费者1

```
//模拟工作队列消费者1
    @RabbitListener(queues = "simple.queue")
    public void workQueueAMQP(String msg) throws Exception{
        System.out.println("消费者1接收到消息：[" + msg +"]" + LocalTime.now());
        Thread.sleep(20);
    }
```

消费者2

```
//模拟工作队列消费者2
    @RabbitListener(queues = "simple.queue")
    public void workQueueAMQP2(String msg) throws Exception{
        System.err.println("消费者2接收到消息：[" + msg +"]" + LocalTime.now());
        Thread.sleep(200);
    }
```

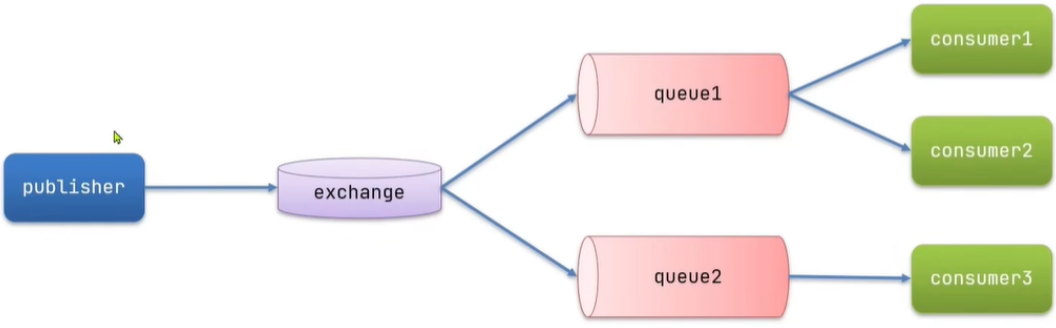
注意：按照上面的实现方式，50条消息会平均分配给两个消费者，这是因为Rabbitmq的消息预取装置，两个队列在消费消息前，会轮流从队列取出消息，不会考虑自身性能

通过在yml文件中修改spring.rabbitmq.listener.direct.prefetch的值来设置预取

```
spring:
  rabbitmq:
    host: 192.168.61.131
    port: 5672
    virtual-host: /
    username: guest
    password: guest
    listener:
      direct:
        prefetch: 1 #每次只能获取一条消息，处理完再进行获取
```

10 SpringAMQP 实现发布订阅模式

发布订阅模式允许将同一条信息发给多个消费者。实现方式是加入exchange(交换机)

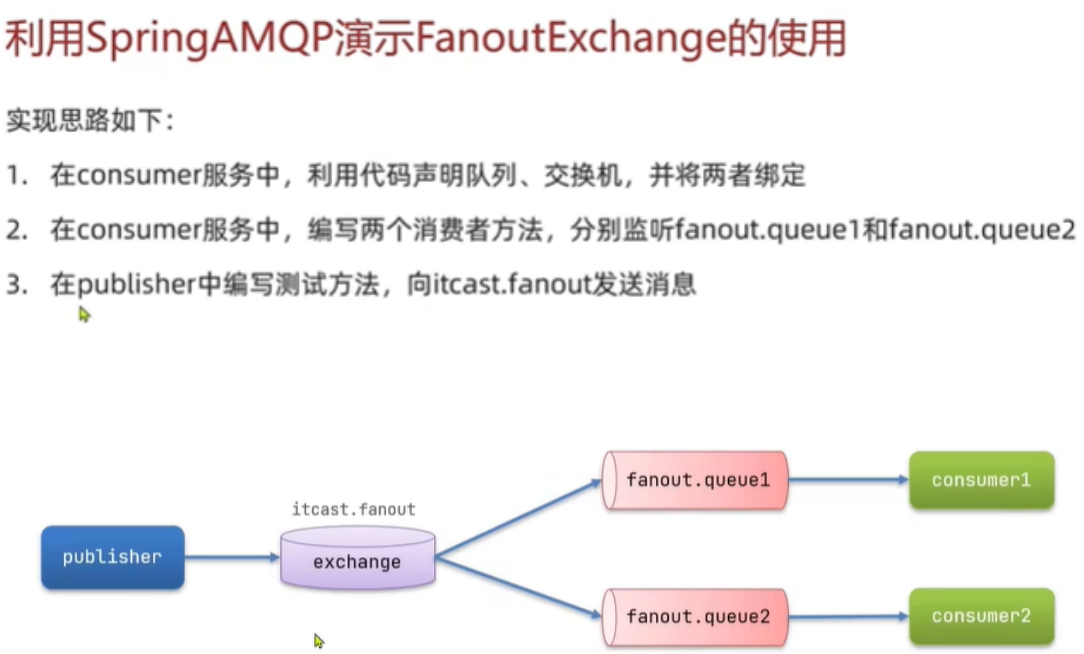


发给几个队列由交换机的类型决定

常见的交换机有：Fanout(广播)、Direct(路由)、Topic(话题)

11 广播交换机（FanoutExchange）

会将接收到的消息路由每个跟其绑定的queue



步骤1：再consumer服务声明ExChange、Queue、Binding

声明交换机

```
//声明交换机
    @Bean
    public FanoutExchange fanoutExchange(){
        return new FanoutExchange("itcast.fanout");

    }
```

声明队列1，并绑定到交换机

```
 //声明队列1
    @Bean
    public Queue fanoutQueue1(){
        return new Queue("fanout.queue1");
    }

    //绑定队列1与交换机
    @Bean
    public Binding fanoutBinding(Queue fanoutQueue1,FanoutExchange
fanoutExchange){
        return BindingBuilder
                .bind(fanoutQueue1)
                .to(fanoutExchange);
    }
```

```
//new Queue，分别是durable,exclusive,autoDelete,
        // durable 即是否持久化.默认是false，
        //exclusive 只能被当前创建的连接使用，而且当连接关闭后队列即被删除.默认也是false
        //是否自动删除，当没有生产者或者消费者使用此队列，该队列会自动删除。
        //设置durable为true其余为false。
```

声明队列2，并绑定到交换机

```
//声明队列2
@Bean
public Queue fanoutQueue2(){
    return new Queue("fanout.queue2");
}

//绑定队列2与交换机
@Bean
public Binding fanoutBinding2(Queue fanoutQueue2,FanoutExchange fanoutExchange){
    return BindingBuilder
            .bind(fanoutQueue2)
            .to(fanoutExchange);
}
```

上面三个步骤均在消费者的配置类下配置

消费者

```
//订阅模式队列1
@RabbitListener(queues = "fanout.queue1")
public void fanoutExchange(String msg) throws Exception{
    System.err.println("fanout.queue1接收到消息：[" + msg +"]" + LocalTime.now());

}

//订阅模式队列2
@RabbitListener(queues = "fanout.queue2")
public void fanoutExchange2(String msg) throws Exception{
    System.err.println("fanout.queue2接收到消息：[" + msg +"]" + LocalTime.now());

}
```

以下代码作用与上面类似

```
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(name = "fanout.queue1"),
        exchange = @Exchange(name = "itcast.fanout",type = ExchangeTypes.FANOUT)
))
public void fanoutExchange3(String msg) throws Exception{
    System.err.println("fanout.queue1接收到消息：[" + msg +"]" + LocalTime.now());

}
```

生产者

```
@Test
public void sendFanoutExchange(){
    //交换机名称
    String exchangeName = "itcast.fanout";

    //消息
    String message = "hello,everyone";

    rabbitTemplate.convertAndSend(exchangeName,"",message);
}
```

交换机的作用：接受生产者发送的消息

将消息按照规则路由到与之绑定的队列

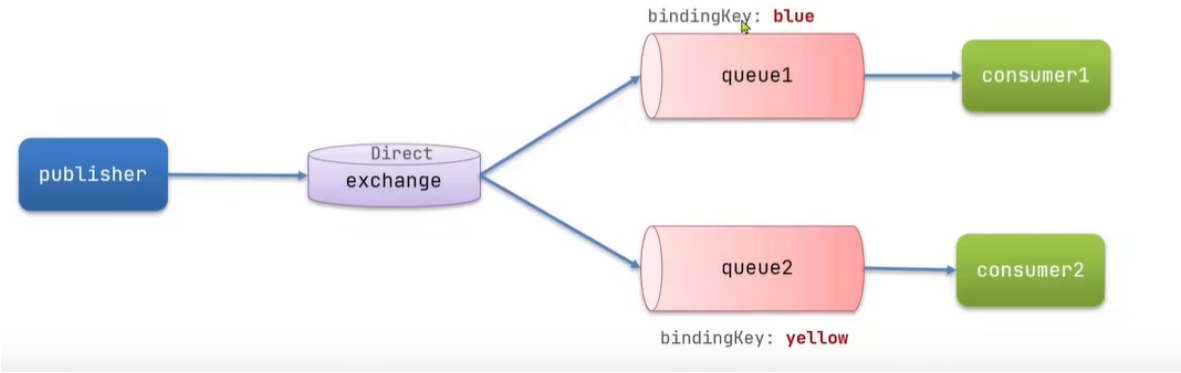
注意：交换机只能做消息的转发不能做消息的存储，如果路由没有成功，消息会丢失

12 路由交换机（Direct）

接受到的消息根据规则路由到指定的队列中（路由模式）

路由规则

- 每一个Queue都与Exchange设置一个BindingKey
- 发布者发送消息时，指定消息的RoutingKey
- Exchange将消息路由到BindingKey与消息RoutingKey一致的队列



注意：一个队列可以绑定多个bindingkey

消费者

```
//路由模式
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(name = "direct.queue1"),
        exchange = @Exchange(name = "itcast.direct",type =
ExchangeTypes.DIRECT),//type可省略
        key = {"red","blue"}
))
public void directQueue1(String msg) throws Exception{
    System.err.println("directQueue接收到消息：[" + msg +"]");

}
```

```
//路由模式2
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(name = "direct.queue2"),
        exchange = @Exchange(name = "itcast.direct",type =
ExchangeTypes.DIRECT),//type可省略
        key = {"red","yellow"}
))
public void directQueue2(String msg) throws Exception{
    System.err.println("directQueue2接收到消息：[" + msg +"]");

}
```

生产者

blue表示队列要具有bindingkey为blue的才可以收到信息，即队列1

```
@Test
public void sendDirectExchange(){
    //交换机名称
    String exchangeName = "itcast.direct";

    //消息
    String message = "hello,blue";

    rabbitTemplate.convertAndSend(exchangeName,"blue",message);
}
```

13 Direct与Fanout区别

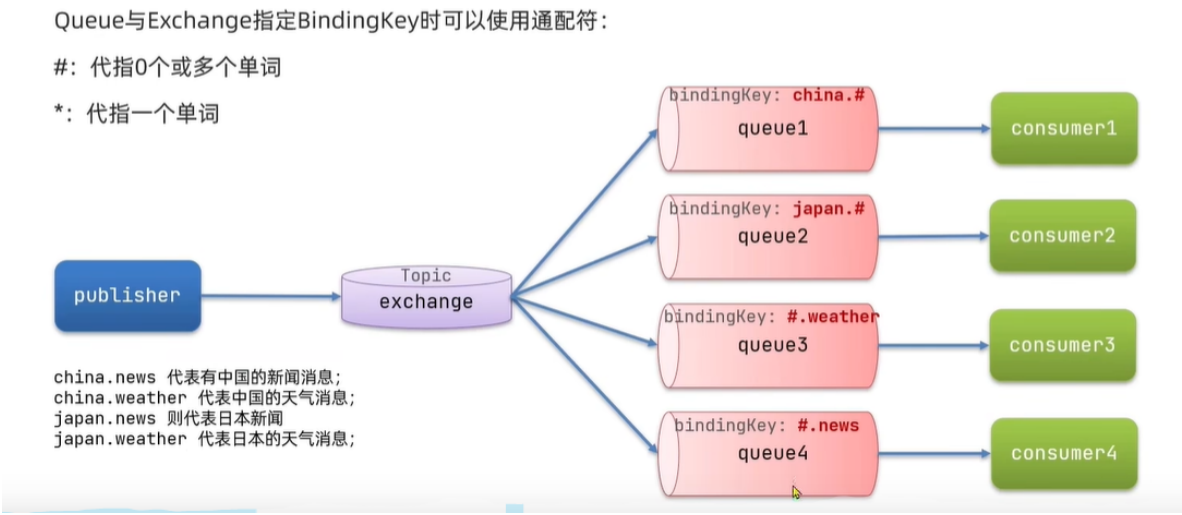
1 Fanout将消息发送给每一个与它绑定的队列

2 direct将消息发送给与Routerkey一致的bindingkey

3 direct中如果多个队列具有相同的bindingkey则功能与fanout相似

14 TopicExchange(发布订阅)

TopicExchange与DirectExchange类似，区别在于Routerkey必须是多个单词的列表，并以点（.）进行分割



实现TopicExchange步骤

消费者

```
//发布订阅模式1
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(name = "topic.queue1"),
        exchange = @Exchange(name = "itcast.topic",type = ExchangeTypes.TOPIC),
        key = "ch.#"
))
public void topicQueue1(String msg) throws Exception{
    System.out.println("消费者topic1接收到消息：[" + msg +"]");
}

//发布订阅模式2
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(name = "topic.queue2"),
        exchange = @Exchange(name = "itcast.topic",type = ExchangeTypes.TOPIC),
        key = "#.news"
))
public void topicQueue2(String msg) throws Exception{
    System.out.println("消费者topic2接收到消息：[" + msg +"]");
}
```

生产者

```
@Test
public void sendTopicExchange(){
    //交换机名称
    String exchangeName = "itcast.topic";

    //消息
    String message = "Rabbitmq学习了";

    rabbitTemplate.convertAndSend(exchangeName,"ch.news",message);
}
```

15 SpringAMQP消息转换器（序列化）

> 说明：在SpringAMQP的发送方法中，接收消息的类型是Object，也就是说我们可以发送任意对象类型的消息，SpringAMQP会帮我们序列化为字节后发送。

声明队列（在消费者的配置类下声明，再重启消费者即可创建成功）

```
@Bean
public Queue objectQueue(){
    return new Queue("object.queue");
}
```

生产者

```
@Test
public void sendObjectMessage(){

    Map<String,Object> msg = new HashMap<>();
    msg.put("name","张三");
    msg.put("age",20);

    rabbitTemplate.convertAndSend("object.queue",msg);
}
```

上面操作会序列化，默认会采用jdk的序列化（ObjectOutputStream），性能差、安全性差、序列化长度过长,所以推荐采用Json序列化格式

json序列化步骤

- 我们在publisher服务引入依赖

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.10</version>
</dependency>
```

- 我们在publisher服务声明MessageConverter：

```
@Bean
public MessageConverter jsonMessageConverter(){
    return new Jackson2JsonMessageConverter();
}
```

消费者

我们在consumer服务引入Jackson依赖：

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.9.10</version>
</dependency>
```

我们在consumer服务定义MessageConverter：

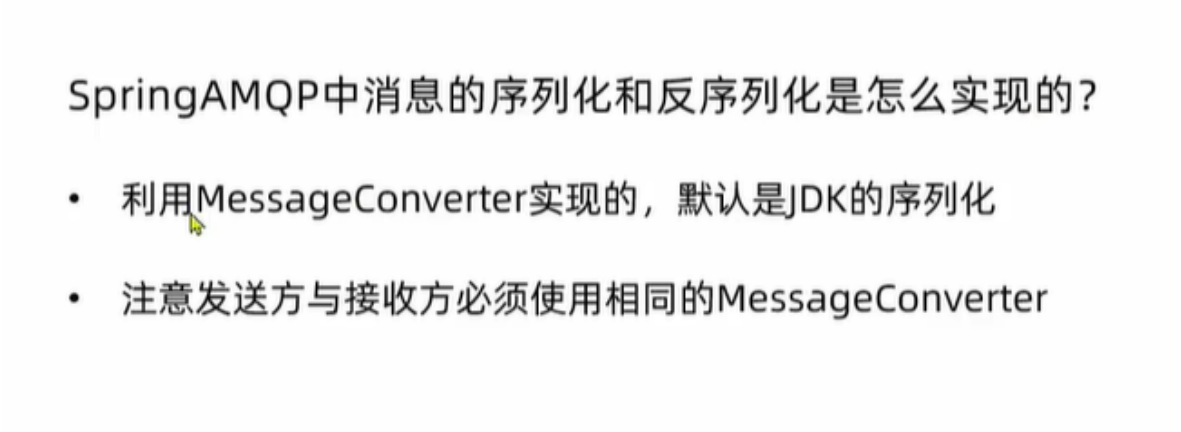
```
@Bean
public MessageConverter jsonMessageConverter(){
    return new Jackson2JsonMessageConverter();
}
```

然后定义一个消费者，监听object.queue队列并消费消息：

```
@RabbitListener(queues = "object.queue")
public void listenObjectQueue(Map<String, Object> msg) {
    System.out.println("收到消息: [" + msg + "] ");
}
```

MessageConverter可以放在启动类上面

序列化与反序列化



16 消息确认ack

消息确认ack：如果在处理消息的过程中，消费者的服务器在处理消息的时候出现异常，那么可能这条正在处理的消息就没有完成消息消费，数据就会丢失。为了确保数据不会丢失，RabbitMQ支持消息确定-ACK。

默认情况下 spring-boot-data-amqp 是自动ACK机制，就意味着 MQ 会在消息发送完毕后，自动帮我们去ACK，然后删除消息的信息。这样依赖就存在这样一个问题：
如果消费者处理消息需要较长时间，最好的做法是消费端处理完之后手动去确认

17 消息的过期时间TTL

过期时间TTL表示可以对消息设置预期的时间，在这个时间内都可以被消费者接收获取；过了之后消息将自动被删除。

注意：删除默认是直接从队列中移除，但一般是将消息移入死信队列

RabbitMQ可以对消息和队列设置TTL。

设置过期时间有两种方式

- 第一种方法是通过队列属性设置，队列中所有消
- 息都有相同的过期时间。
- 第二种方法是对消息进行单独设置，每条消息TTL可以不同。

1 对队列进行设置

声明一个队列交换机，并将二者绑定

```
//队列TTL
@Bean
public Queue ttlQueue(){
    Map<String,Object> args =new HashMap<>();
    args.put("x-message-ttl",5000);//这里一定是int类型
    return new Queue("ttl.queue",true,false,false,args);//将时间参数放上，设置过期时
间
}
//ttl交换机
@Bean
public DirectExchange ttlExchange(){
```

```
    return new DirectExchange("ttl_exchange");
}

//绑定ttl交换机与ttl队列
@Bean
public Binding ttlBinding(){
    return BindingBuilder
            .bind(ttlQueue())
            .to(ttlExchange()).with("ttl");//路由交换机在绑定的时候需要设置Routerkey
}
```

生产者

```
@Test
public void sendDirectExchange(){
    //交换机名称
    String exchangeName = "ttl_exchange";

    //消息
    String message = "hello,ttl";

    rabbitTemplate.convertAndSend(exchangeName,"ttl",message);
}
```

2 给消息设置过期时间

```
//队列TTL
@Bean
public Queue ttlQueue1(){

    return new Queue("ttl.queue",true);
}

//ttl交换机
    @Bean
    public DirectExchange ttlExchange(){
        return new DirectExchange("ttl_exchange");
    }

//绑定ttl交换机与ttl队列
@Bean
public Binding ttlBinding2(){
    return BindingBuilder
            .bind(ttlQueue1())
            .to(ttlExchange()).with("ttl-message");
}
```

在生产者处设置消息的过期时间

```
@Test
    public void ttlMessage(){
```

```
        //交换机名称
        String exchangeName = "ttl_exchange";



        MessagePostProcessor messagePostProcessor = new MessagePostProcessor() {
            @Override
            public Message postProcessMessage(Message message) throws
AmqpException {
                message.getMessageProperties().setExpiration("3000");
                return message;
            }
        };
//        //消息
        String message = "消息设置过期时间";
        rabbitTemplate.convertAndSend(exchangeName,"ttl-
message",message,messagePostProcessor);
    }
```

注意：如果同时对队列和消息设置了过期时间，以ttl值小的为准，比如笔记中消息ttl为3秒，而队列ttl为5秒，则以消的ttl为准

两者区别：对消息设置过期时间，消息一旦过期会直接移除

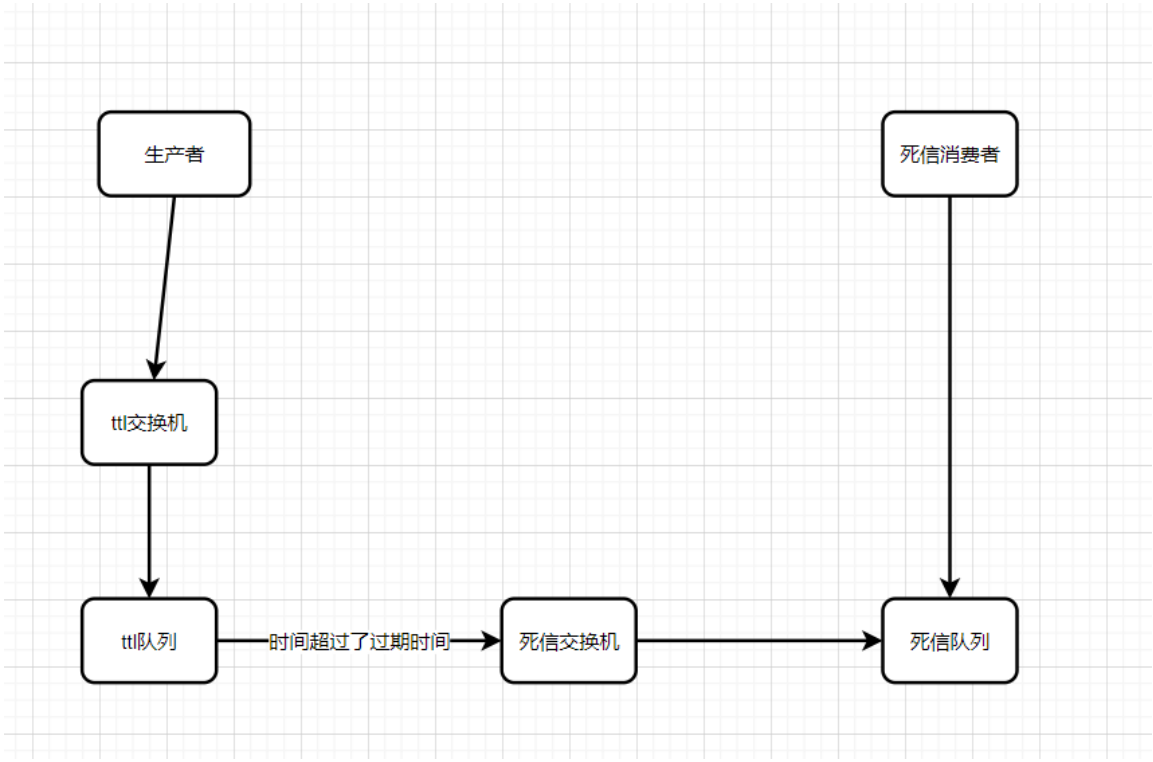
而对队列设置过期时间，则该队列称为过期队列，过期队列的消息过期后可以写入死信队列

18 死信队列

DLX 即死信交换机，当消息在一个队列中变成死信(dead message)之后，它能被重新发送到另一个交换机中，这个交换机就是DLX ，绑定DLX的队列就称之为死信队列。

消息变成死信，可能是由于以下的原因：

- 消息被拒绝
- 消息过期
- 队列达到最大长度



注意：如果队列已经创建了，但是需要修改它的默写参数是需要将队列删除后在进行创建（实际开发中不可以，因该创建新的队列来进行转换和迁移）

创建死信队列

```
@Bean
public DirectExchange deadExchange(){
    return new DirectExchange("dead-exchange");
}

@Bean
public Queue deadQueue(){
    return new Queue("dead.queue");
}

@Bean
public Binding deadBinding(){
    return BindingBuilder.bind(deadQueue())
            .to(deadExchange()).with("dead.queue.exchange");
}
```

在正常队列中设置与死信队列的绑定

```
//队列TTL
@Bean
public Queue ttlQueue(){
    Map<String,Object> args =new HashMap<>();
    args.put("x-message-ttl",5000);//这里一定是int类型，过期时间
    //args.put("x-max-length",3);//设置队列最大长度
    args.put("x-dead-letter-exchange","dead-exchange");//前一个参数为RabbitMq管理界面设置死信队列处的关键字，后一个为死信队列的队列名
```

```
    args.put("x-dead-letter-router-key","ttl");//交换机为Redir交换机才需要设置（即
fanout不需要配置）
    // 前一个参数仍然是从RabbitMq管理界面拿到的，后一个为你设置的Router-key
    return new Queue("ttl.queue",true,false,false,args);//将时间参数放上，设置过期时
间
}
//ttl交换机
@Bean
public DirectExchange ttlExchange(){
    return new DirectExchange("ttl_exchange");
}

//绑定ttl交换机与ttl队列
@Bean
public Binding ttlBinding(){
    return BindingBuilder
            .bind(ttlQueue())
            .to(ttlExchange()).with("ttl");
}
```

过期时间+死信队列=》延迟队列