高并发的核心技术 - 消息中间件（MQ）

2017-08-16

- 什么是MQ
  跨进程的消息队列，主要角色包括生产者与消费者。
  生产者只负责生产信息，无法感知消费者是谁，消息怎么处理，处理结果是什么。
  消费者负责接收及处理消息，无法感知生产者是谁，怎么产生的。

- Mq能做什么?
  MQ 特性一般有异步，吞吐量大 ，延时低;
  适合做:

  1. 投递异步通知。
  2. 限流，削峰谷。
  3. 可靠事件，处理数据一致性。
  4. 利用一些特性，可以做定时任务。
     等….
     *由于MQ是异步处理消息的，所以MQ不适合做同步处理操作，如果需要及时的返回处理结果请不要用MQ;

- MQ 个系统带来了什么?

  缺点：增加了系统的复杂性，除了代码组件接入以外还需要考虑，高可用，集群，消息的可靠性等问题！

  生产者：消息发送怎么保证可靠性，怎么保证不重复！

  消费者：怎么保证幂等性，接收到重复消息怎么处理！

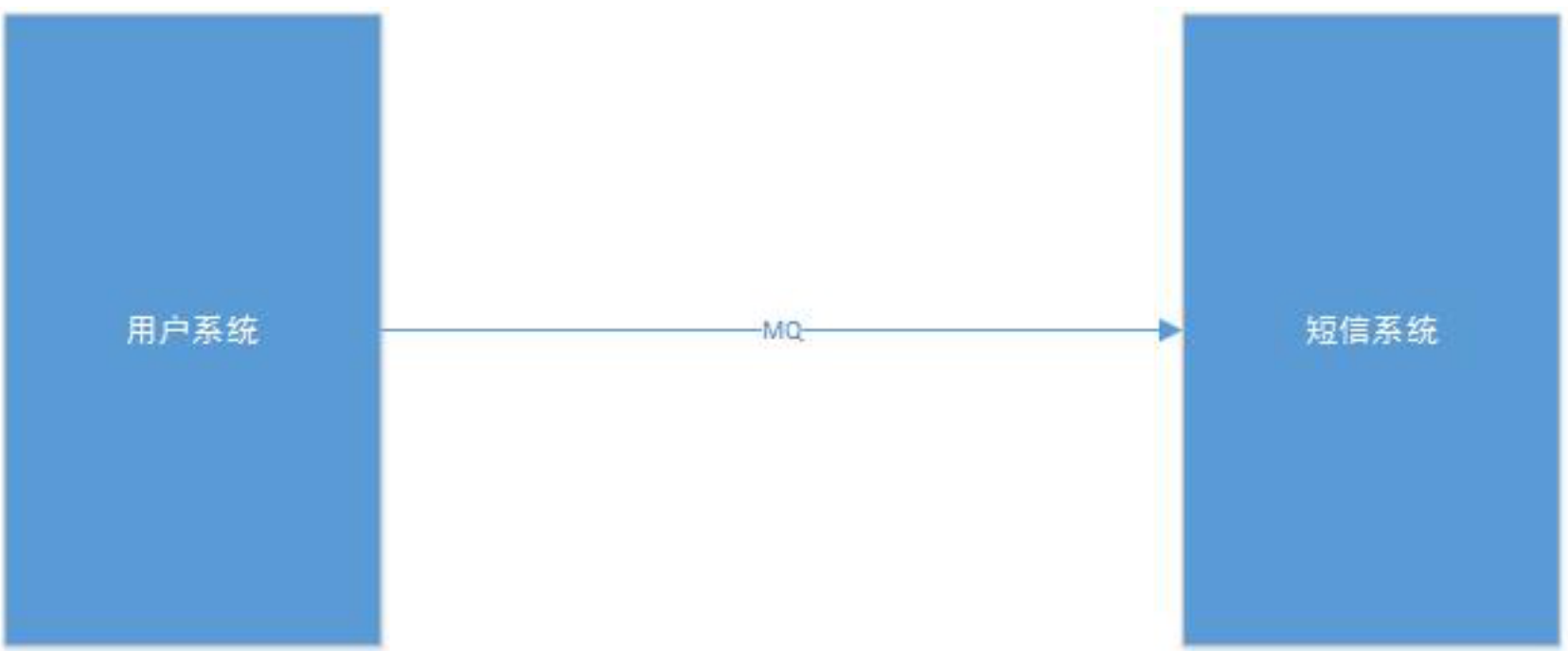
  还有会带来的处理延时等问题!

优点： 解耦，利用MQ我们可以很好的给我们系统解耦，特别是分布式/微服系统！

原来的同步操作，可以用异步处理，也可以带来更快的响应速度；

- 哪些场景可以使用MQ

1. 场景 （1）
   系统解耦，用户系统或者其他系统需要发送短信可以通过 MQ 执行；
   很好的将 用户系统 和 短信系统进行解耦；



1. 场景（2）
   顺序执行的任务场景，假设 A B C 三个任务，B需要等待 A完成才去执行，C需要等待B完成才去执行；

我见过一些同学的做法是 ，用 三个定时器 错开时间去执行的，假设 A定时器 9 点执行， B 定时器 10 点执行 ， C 11 点执行 ， 类似这样子；

这样做其实是 不安全的， 因为 后一个任务 无法知道 前一个任务是否 真的执行了！ 假设 A 宕机了， 到 10 点 B 定时去 执行，这时候 数据就会产生异常 ！

当我们 引入 MQ 后 可以这么做， A执行完了 发送 消息给 B ，B收到消息后执行，C 类似，收到 B消息后执行；

1. 场景（3）
   支付网关的通知，我们的系统常常需要接入支付功能，微信或者支付宝通常会以回调的形式通知我们系统支付结果。
   我们可以将我们的支付网关独立出来，通过MQ通知我们业务系统进行处理，这样处理有利于系统的解耦，

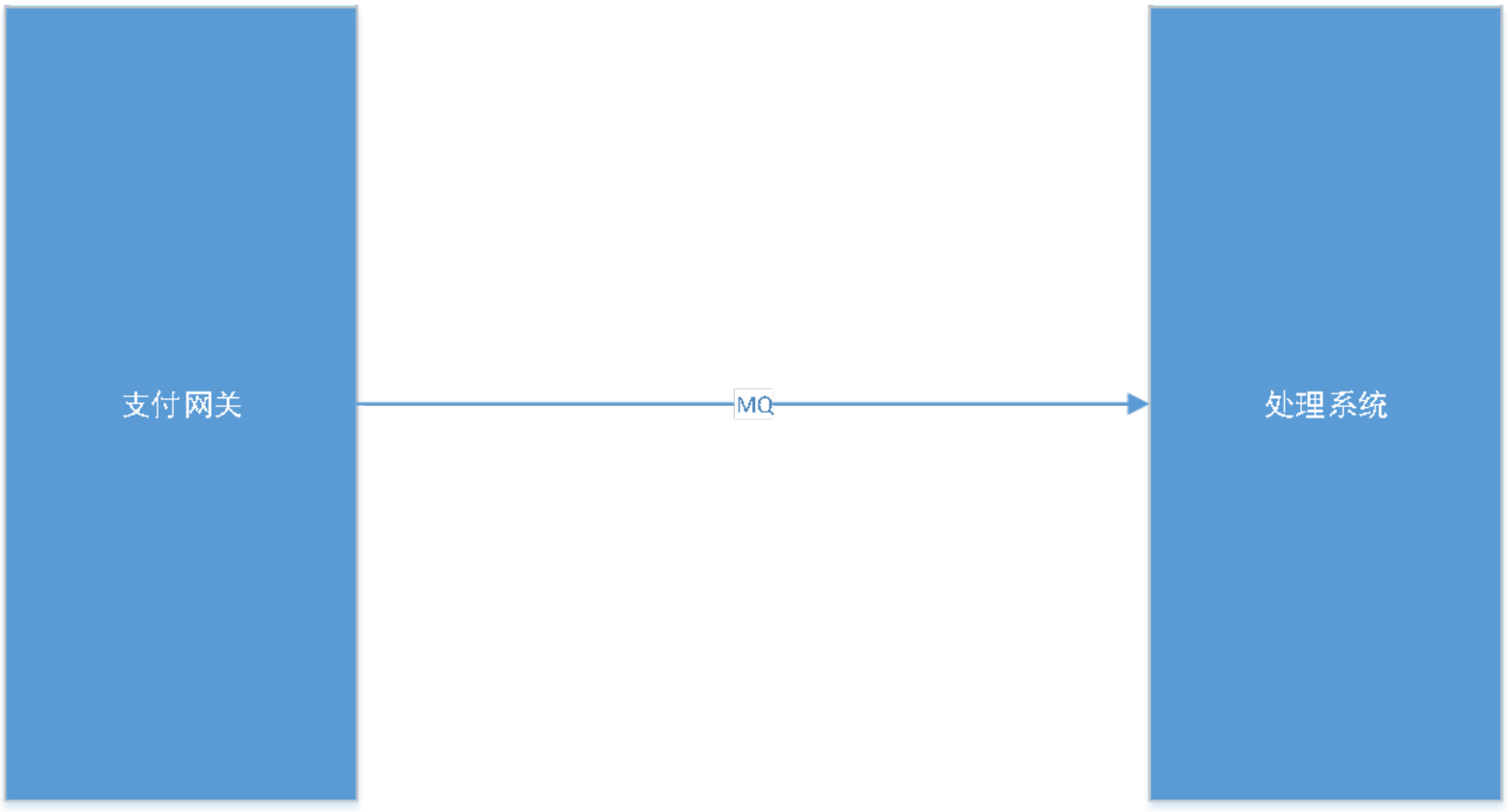
和扩展！

假设我们还有一个积分系统，用户支付成功，给用户添加积分。只需要积分系统监听这个消息，并处理积分就好，无需去修改再去修改网关层代码！

如果没有使用MQ，我是不是还得去修改网关系统的代码，远程调用增加积分的接口？

这就是使用了MQ的好处，解耦和扩展！

当然我们的转发规则也要保证每个感兴趣的队列能获取到消息！



2. 场景（4）

微服/分布式系统，分布式事务 - 最终一致性 处理方案！

详情： http://jblog.top/article/details/257231

1. 场景（5）

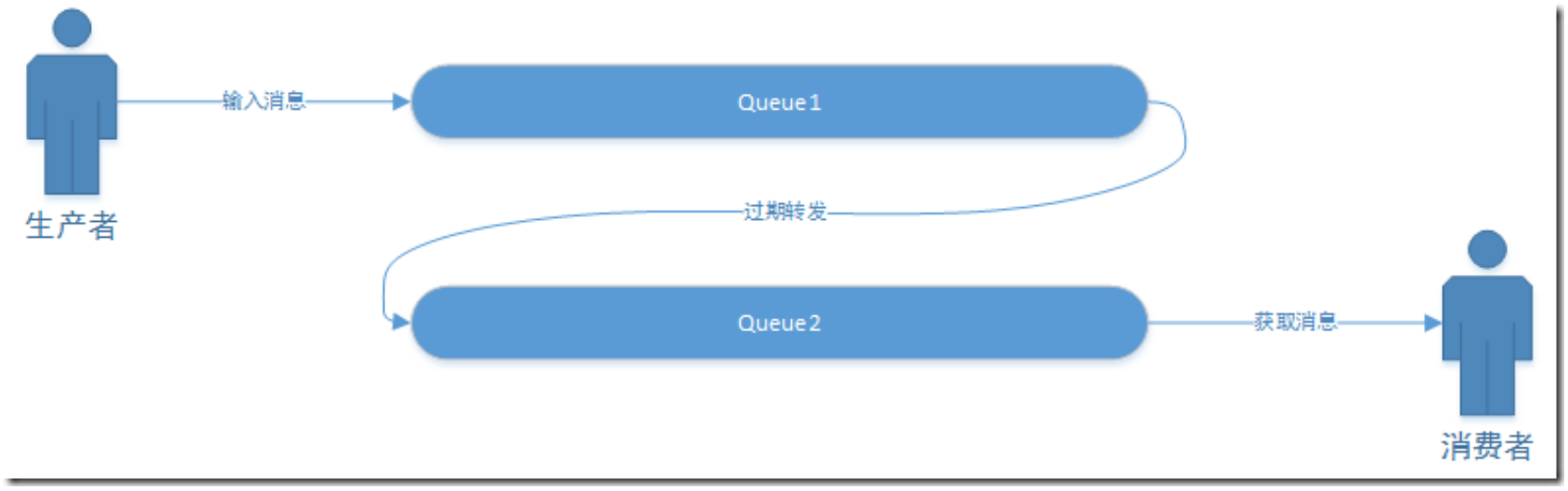
   消息延时队列，可做些定时任务，不固定时间执行的定时任务。
   例如：用户下单后如果24小时未支付订单取消；
   确认收货后2天后没有评价自动好评；
   等

我们以前的做法是 通常启用一个定时器，每分钟或者每小时，去跑一次取出需要处理的订单或其他数据进行处理。
这种做法一个是 效率比较低，如果数据量大的话，每次都要扫库，非常要命！
再者时效性不是很高，最差的时候可能需要等待一轮时长！
还有可能出现重复执行的结果，时效和轮询的频率难以平衡！

利用MQ（Rabbitmq）,DLX （Dead Letter Exchanges）和 消息的 TTL（Time-To-Live Extensions）特性。我们可以高效的完成这个任务场景！不需要扫库，时效性更好！

DLX：http://www.rabbitmq.com/dlx.html,_
TTL：http://www.rabbitmq.com/ttl.html#per-message-ttl

原理：
发送到队列的消息，可以设置一个存活时间 TTL，在存活时间内没有被消费，可以设置这个消息转发到其他队列里面去；然后我们从这个其他队列里面消费执行我们的任务，这样就可以达到一个消息延时的效果！



设置过期时间：
过期时间可以统一设置到消息队列里面，也可以单独设置到某个消息！

*PS 如果消息设置了过期时间，发生到了设置有过期时间的队列，已队列设置的过期时间为准！

已 SpringBoot 为例：

配置转发队列和被转发队列：

```
@Component
@Configuration
public class RabbitMqConfig {

    @Bean
    public Queue curQueue() {
        Map<String, Object> args = new HashMap<String, Object>();
        //超时后的转发器 过期转发到 delay_queue_exchange
        args.put("x-dead-letter-exchange", "delay_queue_exchange");
        //routingKey 转发规则
        args.put("x-dead-letter-routing-key", "user.#");
        //过期时间 20 秒
        args.put("x-message-ttl", 20000);
        return new Queue("cur_queue", false, false, false, args);
    }

    @Bean
    public Queue delayQueue() {
        return new Queue("delay_queue");
    }

    @Bean
    TopicExchange exchange() {
        //当前队列
        return new TopicExchange("cur_queue_exchange");
    }

    @Bean
    TopicExchange exchange2() {
        //被转发的队列
        return new TopicExchange("delay_queue_exchange");
    }

    @Bean
    Binding bindingHelloQueue(Queue curQueue, TopicExchange exchange)
{
         //绑定队列到转发器
        return
BindingBuilder.bind(curQueue).to(exchange).with("user.#");
    }

    @Bean
    Binding bindingHelloQueue2(Queue delayQueue, TopicExchange
exchange2) {
        return
BindingBuilder.bind(delayQueue).to(exchange2).with("user.#");
    }

```

```
45. }
```

发生消息:

```
 1. @Component
 2. public class MqEventSender {
 3.
 4.     Logger logger = LoggerFactory.getLogger(MqEventSender.class);
 5.
 6.     @Autowired
 7.     private RabbitTemplate rabbitTemplate;
 8.
 9.     /**
10.      * 消息没有设置 时间
11.      *  发生到队列 cur_queue_exchange
12.      * @param msg
13.      */
14.
15.     public void sendMsg(String msg) {
16.         logger.info("发送消息:  " + msg);
17.         rabbitTemplate.convertAndSend("cur_queue_exchange", "user.ss",
    msg);
18.     }
19.
20.     /**
21.      * 消息设置时间
22.      *  发生到队列 cur_queue_exchange
23.      * @param msg
24.      */
25.
26.     public void sendMsgWithTime(String msg) {
27.         logger.info("发送消息:  " + msg);
28.         MessageProperties messageProperties = new MessageProperties();
29.         //过期时间设置 10 秒
30.         messageProperties.setExpiration("10000");
31.         Message message =
    rabbitTemplate.getMessageConverter().toMessage(msg,
    messageProperties);
32.         rabbitTemplate.convertAndSend("cur_queue_exchange", "user.ss",
    message);
33.     }
34.
35. }
```

消息监听:

监听 的队列是 delay_queue 而不是 cur_queue；

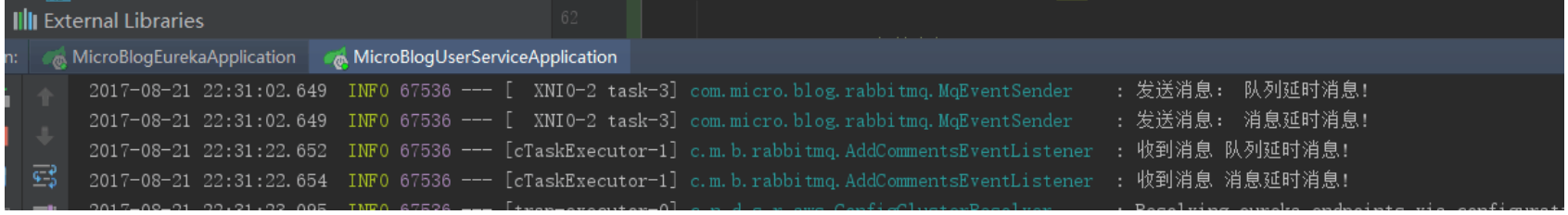
*PS cur_queue 不应该有监听者，否则消息被消费达不到想要的延时消息效果！

```
/**
 * Created by linli on 2017/8/21.
 * 监听 被丢到 超时队列内容
 */
@Component
@RabbitListener(queues = "delay_queue")
public class DelayQueueListener {
    public static Logger logger =
LoggerFactory.getLogger(AddCommentsEventListener.class);

    @RabbitHandler
    public void process(@Payload String msg) {
        logger.info("收到消息 "+msg);
    }

}
```

测试：

```
/**
 * Created by linli on 2017/8/21.
 */
@RestController
@RequestMapping("/test")
public class TestContorller {
    @Autowired
    MqEventSender sender;

    @RequestMapping("/mq/delay")
    public String test() {
        sender.sendMsg("队列延时消息!");
        sender.sendMsgWithTime("消息延时消息!");
        return "";
    }
}
```

结果：



观察结果发现：发送时间 和 收到时间 间隔 20秒 ；

我们给消息设置的 10 秒 TTL 时间没有生效！验证了： 如果消息设置了过期时间，发生到了设置有过期时间的队列，已队列设置的过期时间为准！

如果希望每个消息都要自己的存活时间，发送到队列 不要设置

args.put(“x-message-ttl”, 20000);

消息的过期时间 设置在队列还是消息，根据自己的业务场景去定！

- 总结

MQ 是一个跨进程的消息队列，我们可以很好的利用他进行系统的解耦；
引入MQ会给系统带来一定的复杂度，需要评估！
MQ 适合做异步任务，不适合做同步任务！