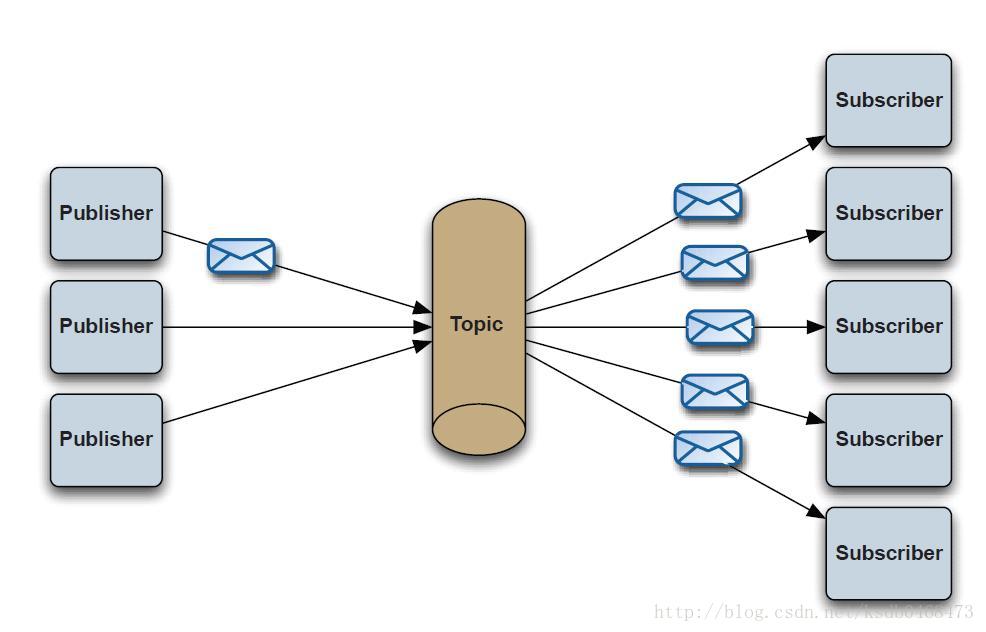
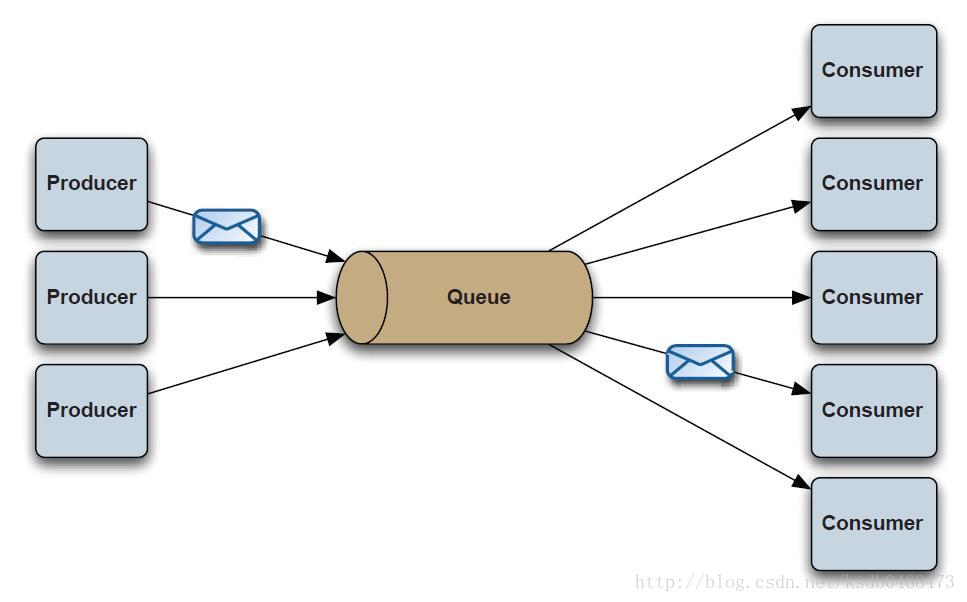
<http://blog.csdn.net/ksdb0468473/article/details/72829700>

要说明如何实现MQ之前，需要先说明一下MQ的分类，总共分为两类：

publish-subscribe

发布订阅模式有点类似于我们日常生活中订阅报纸。每年到年尾的时候，邮局就会发一本报纸集合让我们来选择订阅哪一个。在这个表里头列了所有出版发行的报纸，那么对于我们每一个订阅者来说，我们可以选择一份或者多份报纸。比如北京日报、潇湘晨报等。那么这些个我们订阅的报纸，就相当于发布订阅模式里的topic。有很多个人订阅报纸，也有人可能和我订阅了相同的报纸。那么，在这里，相当于我们在同一个topic里注册了。对于一份报纸发行方来说，它和所有的订阅者就构成了一个1对多的关系。这种关系如下图所示：   


Producer-Consumer

Producer-Consumer的过程则理解起来更加简单。它好比是两个人打电话，这两个人是独享这一条通信链路的。一方发送消息，另外一方接收，就这么简单。在实际应用中因为有多个用户对使用p2p的链路，它的通信场景如下图所示：   


Redis中的publish-subscribe

redis中已经实现了publish-subscribe，订阅者（Subscriber）可以订阅自己感兴趣的频道（Channel），发布者（Publisher）可以将消息发往指定的频道（Channel），正式通过这种方式，可以将消息的发送者和接收者解耦。另外，由于可以动态的Subscribe和Unsubscribe，也可以提高系统的灵活性和可扩展性。   
打开redis客户端，使用SUBSCRIBE命令就可以订阅消息了，如：

SUBSCRIBE china hongkong

发布命令如下：

PUBLISH china "hahahaha"

这样在消息订阅的一方就可以接收到消息了，如下：

1) "message"

2) "china"

3) "hahahaha"

要想取消订阅可以使用：

UNSUBSCRIBE china hongkong

* 1

上面是如何使用redis客户端进行消息的订阅和发布，下面介绍一下如何使用代码实现，我们目前使用Spring Boot的工程框架，所以很多东西不需要手工去配置了，默认Spring Boot会帮我们实现RedisTemplate的bean，所以我们直接注入使用即可。

@Bean

RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,

MessageListenerAdapter listenerAdapter) {

RedisMessageListenerContainer container = new RedisMessageListenerContainer();

container.setConnectionFactory(connectionFactory);

container.addMessageListener(listenerAdapter, new PatternTopic("chat"));

return container;

}

@Bean

MessageListenerAdapter listenerAdapter(Receiver receiver) {

return new MessageListenerAdapter(receiver, "receiveMessage");

}

这里的代码的意思是将消息接收的处理方法和我们的redis订阅端进行一个连接。

return new MessageListenerAdapter(receiver, "receiveMessage");

这里就是接收消息的对象和方法，以后要扩展的话，可以做一个接口，可能通过不同的tag或者是其他的标志，来使用不同的对象处理消息。

container.addMessageListener(listenerAdapter, new PatternTopic("chat"));

代码这里也可以做成接收多个消息的topic，也是需要重构代码的。   
使用RedisTemplate的convertAndSend方法就可以发送消息了，如下：

redisTemplate.convertAndSend("chat", "Hello from Redis!");

* 1

至此，redis的消息发布订阅就介绍完了

Redis中的Producer-Consumer

对于如何实现Producer-Consumer，redis并没有比较直接的方案，但是在list中提供了一个方法RPOPLPUSH，其中官方的资料是这样介绍的：

模式：安全的队列   
Redis通常都被用做一个处理各种后台工作或消息任务的消息服务器。 一个简单的队列模式就是：生产者把消息放入一个列表中，等待消息的消费者用 RPOP 命令（用轮询方式）， 或者用 BRPOP 命令（如果客户端使用阻塞操作会更好）来得到这个消息。   
然而，因为消息有可能会丢失，所以这种队列并是不安全的。例如，当接收到消息后，出现了网络问题或者消费者端崩溃了， 那么这个消息就丢失了。   
RPOPLPUSH (或者其阻塞版本的 BRPOPLPUSH） 提供了一种方法来避免这个问题：消费者端取到消息的同时把该消息放入一个正在处理中的列表。 当消息被处理了之后，该命令会使用 LREM 命令来移除正在处理中列表中的对应消息。   
另外，可以添加一个客户端来监控这个正在处理中列表，如果有某些消息已经在这个列表中存在很长时间了（即超过一定的处理时限）， 那么这个客户端会把这些超时消息重新加入到队列中。

首先说明了，为什么会有这个命令，就是因为在使用RPOP或者BRPOP命令的时候，会出现丢失的问题，所以需要在从一个队列弹出的时候立马将这个对象放到工作队列中，等完成之后再进行删除操作。

在实际的使用中，我们使用的是RPOPLPUSH的阻塞版，也就是说，在没有获取到消息的时候，这个获取的任务会一直阻塞在线程中，直到从队列中取出消息为止。

到目前为止，已经将理论介绍完毕了，下面就说说代码是如何实现的。

String recieveQueueMessage = redisTemplate.opsForList().rightPopAndLeftPush(waitQueue, workQueue, 0, TimeUnit.MILLISECONDS);

* 1

这是最核心的代码部分，使用的是RedisTemplate中用来操作list的接口rightPopAndLeftPush，他是将waitQueue列表最底部的信息弹出，推送到workQueue顶部，等待执行，如果执行都没有问题，再使用

redisTemplate.opsForList().remove(workQueue, REMOVE\_COUNT, messageQueueEntity);

* 1

代码进行删除工作队列的操作，如果没有弹出信息，则继续进行等待，第一个参数是要移出的队列，第二个参数是移出的数目，第三个参数是要移出的内容。

那整体是如何进行工作的呢，下面贴一下整体的代码，然后再详细的进行说明：

@PostConstruct

public void init() {

executorService = Executors.newFixedThreadPool(threadCount);

LOGGER.info("INIT|RECIEVE|MESSAGE|START...");

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

executorService.execute(() -> {

String threadName = Thread.currentThread().getName();

while(true) {

MessageQueueEntity message = channelAdapter.getMessage();

LOGGER.info("RECIEVE|MESSAGE|SUCCESS|{}|{}|", threadName, message);

LOGGER.info("START|HANDLE|MESSAGE|{}", message.getId());

try{

smsSendService.sendSms(message);

} catch(SmsSendErrorException e) {

LOGGER.error("SENDSMS|ERROR|{}|{}", message.getId(), e);

} catch(Exception e) {

e.printStackTrace();

LOGGER.error("SENDSMS|UNKNOW|ERROR|{}|{}", message.getId(), e);

}

LOGGER.info("FINISH|HANDLE|MESSAGE|{}", message.getId());

}

}

);

}

}

@PreDestroy

public void destroy() {

executorService.shutdown();

LOGGER.info("SHUTDOWN|RECIEVE|MESSAGE|SUCCESS|");

}

1. 可以看到使用了spring注解@PostConstruct和@PreDestroy，@PostConstruct注解是要在bean注入的时候去初始化的方法上的，所以当bean进行spring的注入之后，里面的内容就会自动的执行，因为我们要接收信息的时机必须是在启动服务器之后自动就执行，所以使用了这两个注解。
2. 使用了Executors.newFixedThreadPool(threadCount)多线程，这里是固定产生threadCount个线程的线程池，无论是否使用，线程都会等待在那里，threadCount是根据配置来生成了，为了以后能够进行很好的扩展。
3. for(int i = 0; i < threadCount; i++)这里的循环是有几个线程就要执行几次。
4. 后面是比较核心的部分，while（true）可以保证在服务器启动到结束这之间，这几个线程一直在运行，并接收着信息。
5. 接收之后就是之前讲过的使用redis的方式来进行队列的操作
6. 这里值得一提的是，无论多少个线程，多少个消息，他们都是轮询的。

版权声明：本文为博主原创文章，未经博主允许不得转载。