首先我们先说一下Redis当作队列是否合适？

有些人表示赞成，认为Redis很轻量，用作队列很方便

有些人认为Redis会[丢]数据，最好还是使用专业的队列中间件比较合适

1、List队列 如果你的业务需求足够简单，想把Redis当作队列来使用，首先想到的是List这个数据类型，因为List是一个[链表]，在头部和尾部操作元素，时间复杂度都是O(1),这意味着它非常符合，如果把List当作队列，这样

生产者使用LPUSH发布消息 LPUSH queue msg1

消费者这一侧使用RPOP拉取消息 RPOP queue “msg1”

这个很容易理解，但是这里有一个小问题，当队列中没有消息了，消费者在执行PROP时，会返回null，而我们在写消费者逻辑时，这个逻辑一般需要不断的从队列中拉取消息进行处理，伪代码：

while true:

msg = redis.rpop("queue")

// 没有消息，继续循环

if msg == null:

continue

// 处理消息

handle(msg)

如果此时，队列为空，那消费者依旧会频繁的拉取消息，这会造成[CPU空转]，不仅浪费CPU资源，还会对Redis造成压力。我们可以改造一下上边的伪代码，当队列为空时，我们可以[休眠]一会，再去尝试拉取消息，代码如下：

while true:

msg = redis.rpop("queue")

// 没有消息，休眠2s

if msg == null:

sleep(2)

continue

// 处理消息

handle(msg)

这样虽然解决了这个问题，但是会有一个这样的情况，有新的消息进来了，但此时消费者在处理新消息就会有延迟，假设设置的休眠时间是2S，那么信息就会存在2S的延迟，想要缩短这个延迟，只能减少休眠时间，但休眠时间越少，造成CPU空转浪费资源的问题就会出现的越频繁。

此时Redis提供了一种[阻塞式]拉取消息命令，BRPOP/BLPOP，这里B指的就是阻塞(Block)，生产者LPUSH进入队列时，消费者BRPOP，没消息，阻塞等待。伪代码：

while true:

// 没消息阻塞等待，0表示不设置超时时间

msg = redis.brpop("queue", 0)

if msg == null:

continue

// 处理消息

handle(msg)

使用BRPOP这种阻塞方式拉取消息，还支持传入一个[超时时间]，如果设置为0，则表示不设置超时，知道有新消息才返回，否则会在指定的超时时间后返回null。

注意： 如果设置的超时时间太长，这个连接太久没有活跃过，可能会被RedisServer判断为无效连接，之后RedisServer会强制把这个客户端踢下线，所以采用这种方案，客户端要有重连机制。

List作为队列，会存在这两个问题

1：不支持重复消费：消费者拉取消息后，这条消息就从List中删除了，无法被其他消费者再次消费，既不支持多个消费者消费同一批数据

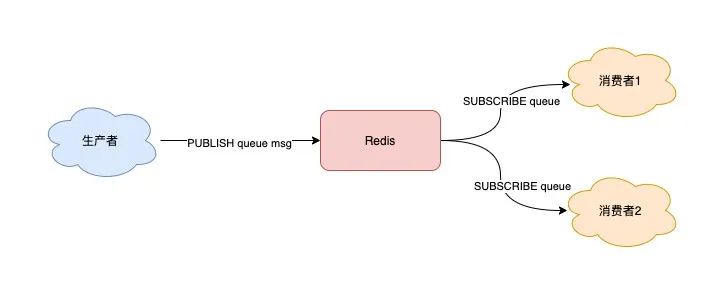
2：消息丢失：消费者拉取消息后，如果发生异常宕机了，那这条数据就丢失了

第一个问题，我们使用List作为消息队列，本身就是功能上的选择，他仅仅支持最简单的，一组生产者对应一组消费者，不能满足多组生产者和消费者的业务场景。

第二个问题，因为从List中拉取一条消息出来后，这条消息就立即从链表中删除了，也就是说无论消费者处理是否成功，这条消息都没法再次消费了。

2、发布/订阅模型：Pub/Sub

这种模式可以解决第一个问题：重复消费，即多组生产者、消费者的场景，



假设你想开启两个消费者，同时消费一批数据，就可以使用SUBSCRIBE 命令启动两个消费者。// 2个消费者 都订阅一个队列

SUBSCRIBE queue

Reading messages... (press Ctrl-C to quit)

1) "subscribe"

2) "queue"

3) (integer) 1

此时，2个消费者都会被阻塞住，等待新消息的到来。

之后启动一个生产者，发布一条新消息。

PUBLISH queue msg1

这时，2个消费者就会解除阻塞，收到生产者发来的新消息。

SUBSCRIBE queue

// 收到新消息

1) "message"

2) "queue"

3) "msg1"

使用Pub/Sub这种方案，即支持阻塞式拉取消息，还能很好的满足了多组消费者，消费同一批数据的业务需求。