链滴

Redis 发布订阅，小功能大用处，真没那么废材！

作者：9526xu

原文链接：https://ld246.com/article/1600357632132

来源网站：链滴

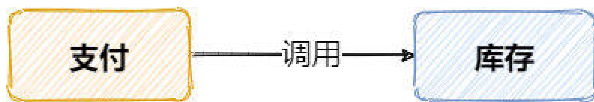
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

今天小黑哥来跟大家介绍一下 Redis 发布/订阅功能。

也许有的小伙伴对这个功能比较陌生，不太清楚这个功能是干什么的，没关系小黑哥先来举个例子。

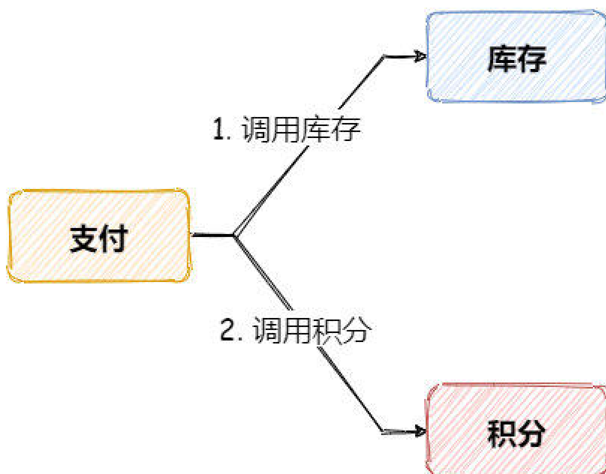
假设我们有这么一个业务场景，在网站下单支付以后，需要通知库存服务进行发货处理。

上面业务实现不难，我们只要让库存服务提供给相关的给口，下单支付之后只要调用库存服务即可。



后面如果又有新的业务，比如说积分服务，他需要获取下单支付的结果，然后增加用户的积分。

这个实现也不难，让积分服务同样提供一个接口，下单支付之后只要调用库存服务即可。



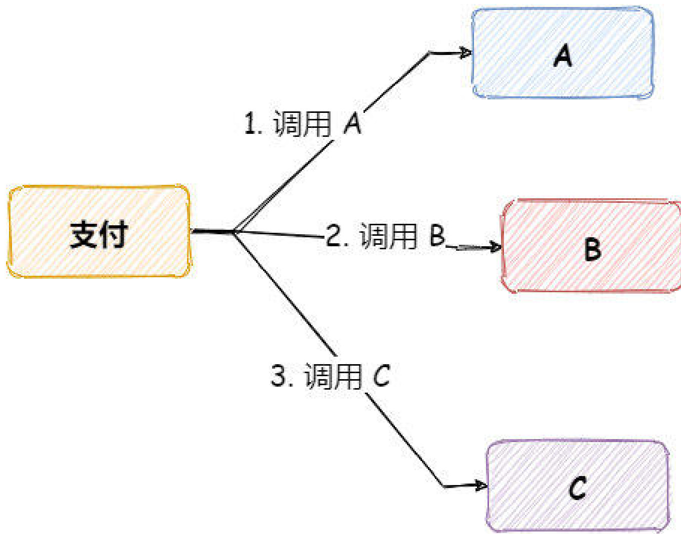
如果就两个业务需要获取下单支付的结果，那也还好，程序改造也快。可是随着业务不断的发展，越越多的新业务说是要下单支付的结果。

这时我们会发现上面这样的系统架构存在很多问题：

第一，下单支付业务与其他业务重度耦合，每当有个新业务需要支付结果，就需要改动下单支付的业
。

第二，如果调用业务过多，会导致下单支付接口响应时间变长。另外，如果有任一下游接口响应变慢就会同步导致下单支付接口响应也变长。

第三，如果任一下游接口失败，可能导致数据不一致的情况。比如说下图，先调用 A，成功之后再调用 B，最后再调用 C。



如果在调用 B 接口的发生异常，此时可能就导致下单支付接口返回失败，但是此时 A 接口其实已经用成功，这就代表它内部已经处理下单支付成功的结果。

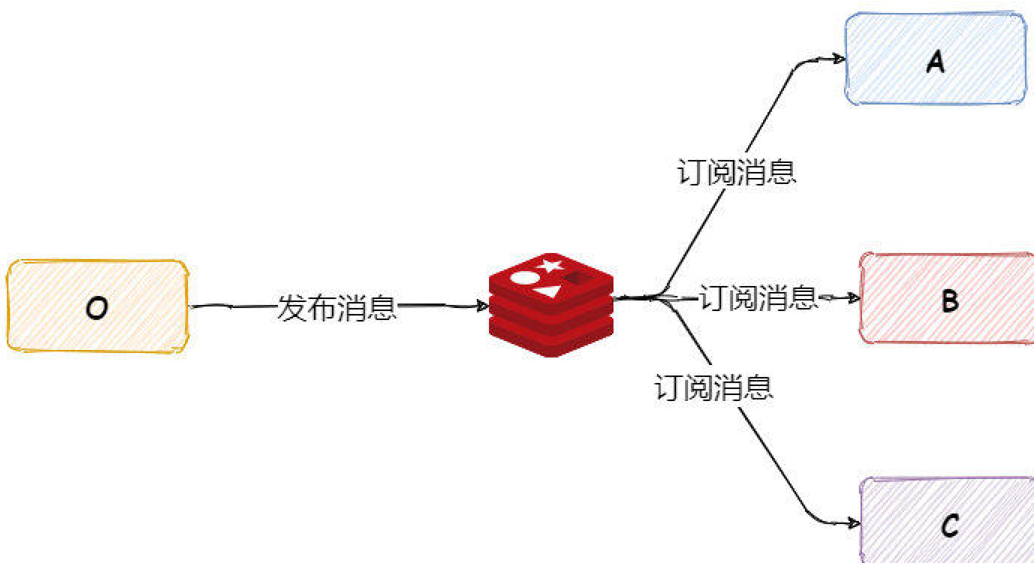
这样就会导致 A，B，C 三个下游接口，A 获取成功获取支付结果，但是 B，C 没有拿到，导致三者统数据不一致的情况。

其实我们仔细想一下，对于下单支付业务来讲，它其实不需要关心下游调用结果，只要有某种机制通能通知到他们就可以了。

讲到这里，这就需要引入今天需要介绍发布订阅机制。

Redis 发布与订阅

Redis 提供了基于「发布/订阅」模式的消息机制，在这种模式下，消息发布者与订阅者不需要进行直通信。



如上图所示，消息发布者只需要想指定的频道发布消息，订阅该频道的每个客户端都可以接受到到这消息。

使用 Redis 发布订阅这种机制，对于上面业务，下单支付业务只需要向**支付结果**这个频道发送消息，

他下游业务订阅**支付结果**这个频道，就能收相应消息，然后做出业务处理即可。

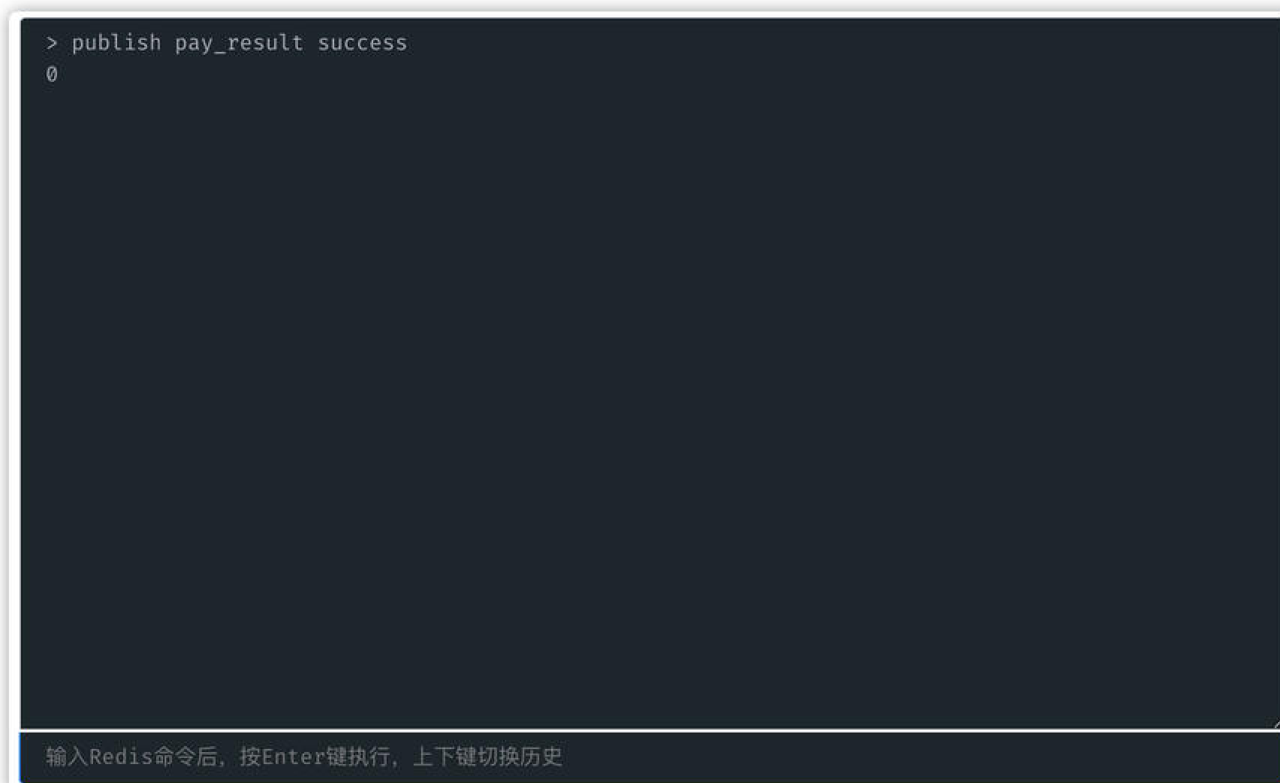
这样就可以解耦系统上下游之间调用关系。

接下来我们来看下，我们来看下如何使用 Redis 发布订阅功能。

Redis 中提供了一组命令，可以用于发布消息，订阅频道，取消订阅以及按照模式订阅。

首先我们来看下如何发布一条消息，其实很简单只要使用 **publish** 指令:

publish channel message



上图中，我们使用 **publish** 指令向 **pay_result** 这个频道发送了一条消息。我们可以看到 redis 向我返回 0 ，这其实代表当前订阅者个数，由于此时没有订阅，所以返回结果为 0 。

接下来我们使用 **subscribe** 订阅一个或多个频道

subscribe channel [channel ...]

原文链接：Redis 发布订阅，小功能大用处，真没那么废材！

```
> subscribe pay_result
1
> get test
Connection in subscriber mode, only subscriber commands may be used
```

```
输入Redis命令后，按Enter键执行，上下键切换历史
```

如上图所示，我们订阅 **pay_result** 这个频道，当有其他客户端往这个频道发送消息，

```
> publish pay_result "message is coming"
1
```

当前订阅者就会收到消息。

```
127.0.0.1:6379> SUBSCRIBE pay_result
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "pay_result"
3) (integer) 1
1) "message"
2) "pay_result"
3) "message is coming"
```

我们子在使用订阅命令，需要主要几点：

第一，客户端执行订阅指令之后，就会进入订阅状态，之后就只能接收 **subscribe**、**psubscribe**、**unsubscribe**、**punsubscribe** 这四个命令。

```
> subscribe pay_result
1
> get test
Connection in subscriber mode, only subscriber commands may be used
> subscribe another_result
2
> unsubscribe pay_result
1
```

第二，新订阅的客户端，是**无法收到这个频道之前的消息**，这是因为 Redis 并不会对发布的消息持久的。

相比于很多专业 MQ，比如 kafka、rocketmq 来说， redis 发布订阅功能就显得有点简陋了。不过 rdis 发布订阅功能胜在简单，如果当前场景可以容忍这些缺点，还是可以选择使用的。

除了上面的功能以外的，Redis 还支持模式匹配的订阅方式。简单来说，客户端可以订阅一个带 * 号模式，如果某些频道的名字与这个模式匹配，那么当其他客户端发送给消息给这些频道时，订阅这个式的客户端也将会到收到消息。

使用 Redis 订阅模式，我们需要使用一个新的指令 **psubscribe**。

我们执行下面这个指令：

psubscribe pay.*

那么一旦有其他客户端往 **pay** 开头的频道，比如 pay_result、pay_xxx，我们都可以收到消息。

```
127.0.0.1:6379> PSUBSCRIBE pay*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "pay*"
3) (integer) 1
1) "pmessage"
2) "pay*"
3) "pay_result"
4) "hello world111"
1) "pmessage"
2) "pay*"
3) "payzxx"
4) "hello world111"
```

如果需要取消订阅模式，我们需要使用相应punsubscribe 指令，比如取消上面订阅的模式：

punsubscribe pay.*

Redis 客户端发布订阅使用方式

基于 Jedis 开发发布/订阅

聊完 Redis 发布订阅指令，我们来看下 Java Redis 客户端如何使用发布订阅。

下面的例子主要基于 Jedis，maven 版本为：

<dependency>

```
 <groupId>redis.clients</groupId>
 <artifactId>jedis</artifactId>
 <version>3.1.0</version>
</dependency>
```

其他 Redis 客户端大同小异。

jedis 发布代码比较简单，只需要调用 Jedis 类的 publish 方法。

```
// 生产环境千万不要这么使用哦，推荐使用 JedisPool 线程池的方式
Jedis jedis = new Jedis("localhost", 6379);
jedis.auth("xxxxx");
jedis.publish("pay_result", "hello world");
```

订阅的代码就相对复杂了，我们需要继承 JedisPubSub 实现里面的相关方法，一旦有其他客户端往阅的频道上发送消息，将会调用 JedisPubSub 相应的方法。

```
private static class MyListener extends JedisPubSub {
    @Override
    public void onMessage(String channel, String message) {
        System.out.println("收到订阅频道：" + channel + " 消息：" + message);

    }

    @Override
    public void onPMessage(String pattern, String channel, String message) {
        System.out.println("收到具体订阅频道：" + channel + "订阅模式：" + pattern + " 消息：" +
message);
    }

}
```

其次我们需要调用 Jedis 类的 subscribe 方法：

```
Jedis jedis = new Jedis("localhost", 6379);
jedis.auth("xxx");
jedis.subscribe(new MyListener(), "pay_result");
```

当有其他客户端往 pay_result频道发送消息时，订阅将会收到消息。

```
Connected to the target VM, address: '127.0.0.1:63770', transport: 'socket'
收到订阅频道: pay_result 消息: hello world
收到订阅频道: pay_result 消息: hello world111
```

不过需要注意的是，jedis#subscribe 是一个阻塞方法，调用之后将会阻塞主线程的，所以如果需要正式项目使用需要使用异步线程运行，这里就不演示具体的代码了。

基于 Spring-Data-Redis 开发发布订阅

原生 jedis 发布订阅操作，相对来说还是有点复杂。现在我们很多应用已经基于 SpringBoot 开发，用 spring-boot-starter-data-redis ，可以简化发布订阅开发。

首先我们需要引入相应的 startter 依赖：

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-redis</artifactId>
   <exclusions>
      <exclusion>
         <artifactId>lettuce-core</artifactId>
         <groupId>io.lettuce</groupId>
      </exclusion>
   </exclusions>
</dependency>
<dependency>
   <groupId>redis.clients</groupId>
   <artifactId>jedis</artifactId>
</dependency>
```

这里我们使用 Jedis 当做底层连接客户端，所以需要排除 lettuce，然后引入 Jedis 依赖。

然后我们需要创建一个消息接收类，里面需要有方法消费消息：

```
@Slf4j
public class Receiver {
   private AtomicInteger counter = new AtomicInteger();

   public void receiveMessage(String message) {
      log.info("Received <" + message + ">");
      counter.incrementAndGet();
   }

   public int getCount() {
      return counter.get();
   }
}
```

接着我们只需要注入 Spring- Redis 相关 Bean，比如：

- StringRedisTemplate，用来操作 Redis 命令
- MessageListenerAdapter ，消息监听器，可以在这个类注入我们上面创建消息接受类 Receiver
- RedisConnectionFactory, 创建 Redis 底层连接

```
@Configuration
public class MessageConfiguration {

   @Bean
   RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,
                                           MessageListenerAdapter listenerAdapter) {

      RedisMessageListenerContainer container = new RedisMessageListenerContainer();
      container.setConnectionFactory(connectionFactory);
      // 订阅指定频道使用 ChannelTopic
      // 订阅模式使用 PatternTopic
      container.addMessageListener(listenerAdapter, new ChannelTopic("pay_result"));
```

```
        return container;
    }

    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        // 注入 Receiver，指定类中的接受方法
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }

    @Bean
    Receiver receiver() {
        return new Receiver();
    }

    @Bean
    StringRedisTemplate template(RedisConnectionFactory connectionFactory) {
        return new StringRedisTemplate(connectionFactory);
    }

}
```

最后我们使用 StringRedisTemplate#convertAndSend 发送消息，同时 Receiver 将会收到一条消
。

```
@SpringBootApplication
public class MessagingRedisApplication {
    public static void main(String[] args) throws InterruptedException {

        ApplicationContext ctx = SpringApplication.run(MessagingRedisApplication.class, args);

        StringRedisTemplate template = ctx.getBean(StringRedisTemplate.class);
        Receiver receiver = ctx.getBean(Receiver.class);

        while (receiver.getCount() == 0) {
            template.convertAndSend("pay_result", "Hello from Redis!");
            Thread.sleep(500L);
        }

        System.exit(0);
    }
}
```

```
2020-09-17 08:28:11.204  INFO 36627 --- [    container-2] com.just.Receiver                        : Received <Hello from Redis!>
```

Redis 发布订阅实际应用

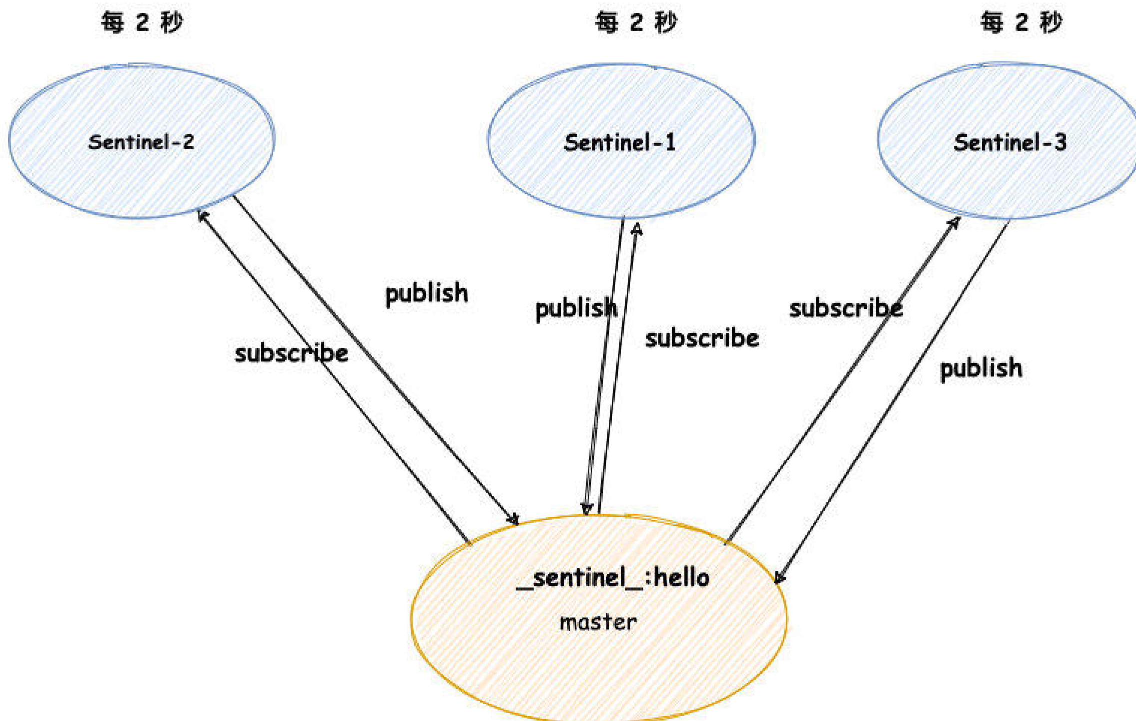
Redis Sentinel 节点发现

Redis Sentinel 是 Redis 一套高可用方案，可以在主节点故障的时候，自动将从节点提升为主节点从而转移故障。

今天这里我们不详细解释 **Redis Sentinel** 详细原理，主要来看下 **Redis Sentinel** 如何使用发布订机制。

Redis Sentinel 节点主要使用发布订阅机制，实现新节点的发现，以及交换主节点的之间的状态。

如下所示，每一个 **Sentinel** 节点将会定时向 _sentinel_:hello 频道发送消息，并且每个 **Sentinel** 会订阅这个节点。



这样一旦有节点往这个频道发送消息，其他节点就可以立刻收到消息。

这样一旦有的新节点加入，它往这个频道发送消息，其他节点收到之后，判断本地列表并没有这个节，于是就可以当做新的节点加入本地节点列表。

除此之外，每次往这个频道发送消息内容可以包含节点的状态信息，这样可以作为后面 **Sentinel** 领者选举的依据。

以上都是对于 Redis 服务端来讲，对于客户端来讲，我们也可以用到发布订阅机制。

当 **Redis Sentinel** 进行主节点故障转移，这个过程各个阶段会通过发布订阅对外提供。

对于我们客户端来讲，比较关心切换之后的主节点，这样我们及时切换主节点的连接（旧节点此时已障，不能再接受操作指令），

客户端可以订阅 +switch-master频道，一旦 **Redis Sentinel** 结束了对主节点的故障转移就会发布节点的的消息。

redission 分布式锁

redission 开源框架提供一些便捷操作 Redis 的方法，其中比较出名的 redission 基于 Redis 的实现布式锁。

今天我们来看下 Redis 的实现分布式锁中如何使用 Redis 发布订阅机制，提高加锁的性能。

PS:redission 分布式锁实现原理，可以参考之前写过的文章:

1. 可重入分布式锁的实现方式
   2. Redis 分布式锁，看似简单，其实真不简单

首先我们来看下 redission　加锁的方法：

```
Redisson redisson = ....
RLock redissonLock = redisson.getLock("xxxx");
redissonLock.lock();
```

RLock 继承自 Java 标准的 Lock 接口,调用 lock 方法,如果当前锁已被其他客户端获取，那么当前加的线程将会被阻塞，直到其他客户端释放这把锁。

这里其实有个问题，当前阻塞的线程如何感知分布式锁已被释放呢?

这里其实有两种实现方法：

第一钟，定时查询分布时锁的状态，一旦查到锁已被释放（Redis 中不存在这个键值），那么就去加。

实现伪码如下：

```
while (true) {
  boolean result=lock();
  if (!result) {
    Thread.sleep(N);
  }
}
```

这种方式实现起来起来简单，不过缺点也比较多。

如果定时任务时间过短，将会导致查询次数过多，其实这些都是无效查询。

如果定时任务休眠时间过长，那又会导致加锁时间过长，导致加锁性能不好。

那么第二种实现方案，就是采用服务通知的机制，当分布式锁被释放之后，客户端可以收到锁释放的息，然后第一时间再去加锁。

这个服务通知的机制我们可以使用 Redis 发布订阅模式。

当线程加锁失败之后，线程将会订阅 redisson_lock__channel_xxx（xx 代表锁的名称） 频道，使用步线程监听消息，然后利用 Java 中 Semaphore 使当前线程进入阻塞。

一旦其他客户端进行解锁，redission 就会往这个redisson_lock__channel_xxx 发送解锁消息。

等异步线程收到消息，将会调用 Semaphore 释放信号量，从而让当前被阻塞的线程唤醒去加锁。

ps：这里只是简单描述了 redission 加锁部分原理，出于篇幅，这里就不再消息解析源码。

感兴趣的小伙伴可以自己看下 redission　加锁的源码。

通过发布订阅机制，被阻塞的线程可以及时被唤醒，减少无效的空转的查询，有效的提高的加锁的效。

ps: 这种方式，性能确实提高，但是实现起来的复杂度也很高，这部分源码有点东西，快看晕了。

总结

今天我们主要介绍 Redis 发布订阅功能，主要对应的 Redis 命令为:

- **subscribe channel [channel ...]** 订阅一个或多个频道
- **unsubscribe channel** 退订指定频道
- **publish channel message** 发送消息
- **psubscribe pattern** 订阅指定模式
- **punsubscribe pattern** 退订指定模式

我们可以利用 Redis 发布订阅功能，实现的简单 MQ 功能，实现上下游的解耦。

不过需要注意了，由于 Redis 发布的消息不会被持久化，这就会导致新订阅的客户端将不会收到历史息。

所以，如果当前的业务场景不能容忍这些缺点，那还是用专业 MQ 吧。

最后介绍了两个使用 Redis 发布订阅功能使用场景供大家参考。