消息发布与订阅

订阅与发布

```
Stream的前置知识：消息发布与订阅
最早Redis就是支持发布和订阅的，通过pub和sub。
进程间的一种消息通信模式：发送者（pub）发送消息，订阅者（sub）接受消息。

消息的发布和订阅，一般是作用在消息中间件的，所以由此可知，Redis不仅想要解决缓存的问题，还想解决通信的问题。
这也是为什么后来Redis推出了Stream，就是像解决这种通信问题的。

* 我们先来看，在没有Stream之前，Redis是怎么解决的：
  * 什么是 “发布订阅”，其实就和我们订阅微信公众号是一样的，
    * 有一个调度中心，
    * 有很多订阅者，
    * 有很多发布者
  * 发布者在调度中心，发布了对应的频道，订阅了这个频道的订阅者，就能够收到消息。（这里用频道来指代他们的对应关系，也可以带入公众号的场景来说，就比如：你订阅了zhoudbw_tian这个公众号，公众号的发布人员其实就是发布者，他会去公众号内发布消息，微信其实本身就是一个调度中心，它会你订阅了这个公众号，然后他会给你推送这个发布的消息，而你就相当于订阅者。）
```

pub / sub 实现

```
Redis提供pub和sub来让我们实现发布与订阅，Redis通过频道来对应：谁在什么频道发送了什么消息，谁又订阅了这个消息。
* 我们以这个订阅和发布的情景来演示命令的使用，
* 对于订阅者和发送者而言，他们都是用户，
* 所以我们演示需要启动两个客户端，
* 下面我用两个框来分别表现两个客户端的现象。
```

```
** 该框用来记录命令：
subscribe channel [channel ...] ：表示该用户订阅了channel，会持续接受消息
publish channel message ：  发布者向channel发布消息为message的消息，相应的订阅者会收到消息
---
命令汇总：
* PSUBSCRIBE pattem [pattem......]：订阅一个或多个符合给定模式的频道
* PUBSUB subcommand [argument [argument.....]]：查看订阅与发布系统状态。
* PUBLISH channel message：将信息发送到指定的频道。
* PUNSUBSCRIBE [pattem [pattem....]]：退订所有给定模式的频道。
* SUBSCRIBE channel [channel...]：订阅给定的一个或多个频道的信息。
* UNSUBSCRIBE [channel [channel...]]：指退订给定的频道。
```

```
** 该框用来描述订阅者：
--order1--
127.0.0.1:6379> subscribe zhoudbw_tian
```

```
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "zhoudbw_tian"
3) (integer) 1
* 这时候，光标会一直在闪烁，等待发布者发布消息
--order3--
* 此时看订阅者那边的变化
127.0.0.1:6379> subscribe zhoudbw_tian
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "zhoudbw_tian"
3) (integer) 1
1) "message"
2) "zhoudbw_tian"
3) "I Love Tian"
* 收到了发布者发布的消息，并且光标还在闪烁，等待消息
```

```
** 该框用来描述发布者：
--order2--
* 订阅者在等待，这时候发布者发布消息
127.0.0.1:6379> publish zhoudbw_tian "I Love Tian"
(integer) 1
127.0.0.1:6379>
```

现在我们知道了什么是发布于订阅，那么我们开始介绍Stream。

扩展数据类型之Stream

说明

```
我们为什么学习Stream？作为Redis 5.0 更新的特性，被重磅推出，Redis开发者也很不讳言的说，Stream极大的借鉴了Kafka的设计。
Stream是Redis 5.0 引入的一种新数据类型，允许消费者等待生产者发送的新数据，还引入了消费者组概念，组之间数据是相同的（前提是设置的偏移量一样），组内的消费者不会拿到相同数据。这种概念和kafka很雷同
由上可知也想要做到和消息中间件一样的功能。Stream比pub和sub的性能、智能更加的强大。
强大主要体现在消息本身的持久化，以及主备复制的功能等等。

我们原本使用的pub和sub也是能够满足消息的传输的，而且逻辑比较的简单，但是它有一个很严重的问题，当Redis重启，或者用户网络中断的时候，那么用户就看不到任何的历史消息，重新连接之后，原先的消息都已经清零了。只有重新去订阅这个频道，才能接受消息，并且这个消息是订阅之后发布的。对于Redis本身重启，这个问题就更加的严重了，所有的用户都需要重新去订阅这个频道，它并没有记录订阅的消息，以及订阅本身的关系，这是很麻烦的一种使用方式。
Stream就解决了上述的问题，Stream支持持久化，而且还支持一种消费者组的概念，这种消费者组（CustomerGroup）的概念，在Kafka中使用的很广泛，所以作者自己也说，是借鉴了Kafka。这样很可以理解，因为技术发展的本质或者说技术的发展过程中也是相互借鉴的过程。

Stream相关的命令都是以x开头的。
* 由于是 5.0 版本之后的，那么原先的 3.2.12，就不支持了，这里做个安装的记录，如下：
```

Redis-5.0.0的安装

```
* 1. 获取tar包：wget http://download.redis.io/releases/redis-5.0.0.tar.gz
  * [root@VM-0-10-centos ~]# wget http://download.redis.io/releases/redis-5.0.0.tar.gz
  * 在当前路径下输入：ll，即可看到：Jun 27  2020 redis-5.0.0.tar.gz
* 2. 解压tar包：tar xzf redis-5.0.0.tar.gz
  * [root@VM-0-10-centos ~]# tar xzf redis-5.0.0.tar.gz
  * 我们来对比一下，现在输入ll，得到的文件：
    drwxrwxr-x 6 root root    4096 Oct 17  2018 redis-5.0.0
    -rw-r--r-- 1 root root 1947721 Jun 27  2020 redis-5.0.0.tar.gz
* 3. 进入解压后的文件夹，输入make编译redis-5.0.0内的所有内容
  * cd redis-5.0.0
  * make
  * 补充make命令的一些知识点：
    make:管理员用它通过命令行来编译和安装很多开源的工具，程序员用它来管理他们大型复杂的项目编译问题

    make 命令像命令行参数一样接收目标。这些目标通常存放在以 “Makefile” 来命名的特殊文件中，同时文件也包含与目标相对应的操作。

    当 make 命令第一次执行时，它扫描 Makefile 找到目标以及其依赖。如果这些依赖自身也是目标，继续为这些依赖扫描 Makefile 建立其依赖关系，然后编译它们。一旦主依赖编译之后，然后就编译主目标（这是通过 make 命令传入的）。

现在，假设你对某个源文件进行了修改，你再次执行 make 命令，它将只编译与该源文件相关的目标文件，因此，编译完最终的可执行文件节省了大量的时间。

* 4. 现在编译的二进制文件可以在src文件夹下得到
  * 运行Redis，用该命令：· src/redis-server ·
  * 使用内置的客户端和Redis交互，用该命令：· src/redis-cli ·
  * [root@VM-0-10-centos ~]# ~/redis-5.0.0/src/redis-cli -p 6379
* 5. end;
```

命令

发布者

```
发布者生成消息：
1. 生成消息，返回ID：xadd streamKeyName ID field string [field string ...]
  如：
        127.0.0.1:6379> xadd zhoudbw_tian * msg I Love Tian
        "1625361702993-0"
        127.0.0.1:6379>
        * 向频道zhoudbw_tian中，添加消息msg=I Love Tian, ID设置为* ，表示让Redis自动设置
        * 返回一个ID，这个ID的解释：“ - ”前是一个时间戳，
          “ - ”后是消息的顺序（顺序指的是该毫秒下产生的第几条消息）。
        127.0.0.1:6379> xadd zhoudbw_tian * msg Forever
```

```
        "1625361869633-0"
        127.0.0.1:6379>
        * 每次生成的时间戳都是不一样的，消息id也是不一样的。

2. 查看当前消息的条数： xlen streamKeyName
  如：
        127.0.0.1:6379> xlen zhoudbw_tian
        (integer) 2

3. 查看当前streamKeyName下的消息内容：xrange streamKeyName start end [COUNT count]
  如：
      127.0.0.1:6379> xrange zhoudbw_tian - +
        1) 1) "1625361702993-0"
           2) 1) "msg"
              2) "I"
              3) "Love"
              4) "Tian"
        2) 1) "1625361869633-0"
           2) 1) "msg"
              2) "Forever"
        127.0.0.1:6379>
        * - + 查看所有的消息的具体内容

4. 删除，给定ID的某条消息：xdel streamKeyName ID [ID ...]，成功返回1，失败0
```

订阅者

```
* 下面开始订阅者，订阅消息(另开一个客户端测试)：
xread [Count count] [BLOCK milliseconds] streams streamKeyName [streamKeyName...]
ID [ID...]
如：
    127.0.0.1:6379> xread streams zhoudbw_tian 0-0
    1) 1) "zhoudbw_tian"
       2) 1) 1) "1625361702993-0"
             2) 1) "msg"
                2) "I"
                3) "Love"
                4) "Tian"
          2) 1) "1625361869633-0"
             2) 1) "msg"
                2) "Forever"
    127.0.0.1:6379>
    * 如果ID不知道可以，使用0-0，读取到给定频道的所有的消息。
-参数，Count count ，如果想要读取指定条数，可以指定count具体值
如：
  127.0.0.1:6379> xread count 1 streams zhoudbw_tian 0-0
    1) 1) "zhoudbw_tian"
       2) 1) 1) "1625361702993-0"
             2) 1) "msg"
```

```
                2) "I"
                3) "Love"
                4) "Tian"
    127.0.0.1:6379>
```
如果想要阻塞的去等待消息，参数block，不能使用0-0了，而是使用$， 在尾部等待消息（效果就像是使用sub/pub的时候，光标一直闪烁等待发布者发布消息）
```订阅者
    127.0.0.1:6379> xread block 0 streams zhoudbw_tian $
    光标闪烁等待消息.
    1) 1) "zhoudbw_tian"
       2) 1) 1) "1625364868575-0"
             2) 1) "msg"
                2) "wating"
    (113.84s)
    127.0.0.1:6379>
    发布者，发布消息结束，订阅者接收到消息之后，也就结束了阻塞，打印出了结果
```
****
```发布者
    127.0.0.1:6379> xadd zhoudbw_tian * msg wating
    "1625364868575-0"
    127.0.0.1:6379>
```

命令汇总：
* 基础操作：xadd / x read
* 范围操作：x range / x revrange
* 消费者操作： xgroup / xack

原理

与redis的 pub / sub不同，pub / sub多个客户端是收到相同的数据，而stream的多个客户端是竞争关系，每个客户端收到的数据是不相同的。