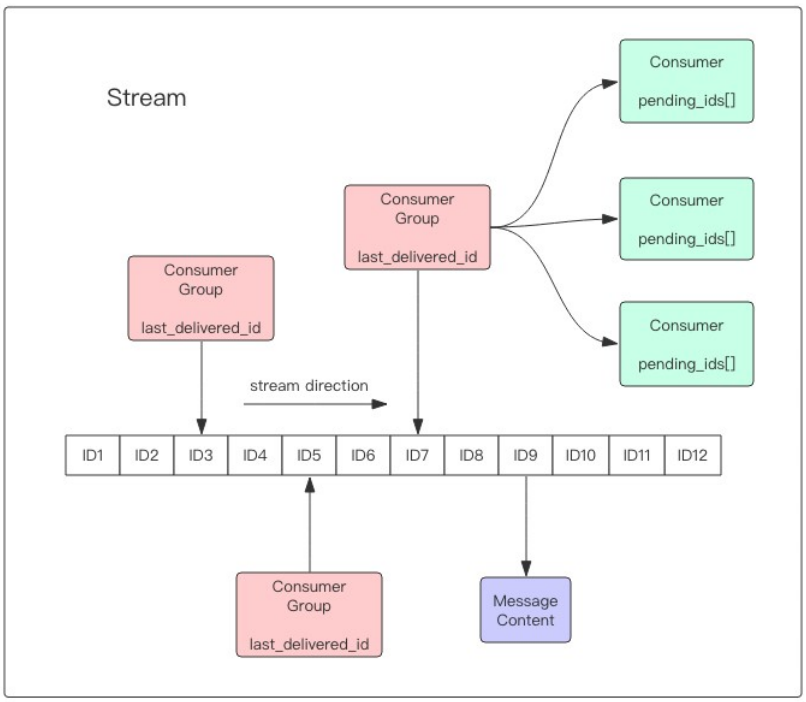
拓展 1：耳听八方 —— Stream

> 这篇文章的主要部分是我在 2018 年 6 月 1 日 发表的内容，当时 Redis5.0 刚刚被 Antirez 放出来，老钱抢先分析了 Redis5.0 最重要的特性 Stream，也被技术圈大号「高可用架构」收录转载，受到了广泛关注。事后我又对文章的内容做了进一步优化，并整合了一些读者的疑难问题解答。

Redis5.0 被作者 Antirez 突然放了出来，增加了很多新的特色功能。而 Redis5.0 最大的新特性就是多出了一个数据结构 **Stream**，它是一个新的强大的支持多播的可持久化的消息队列，作者坦言 Redis Stream 狠狠地借鉴了 Kafka 的设计。



Redis Stream 的结构如上图所示，它有一个消息链表，将所有加入的消息都串起来，每个消息都有一个唯一的 ID 和对应的内容。消息是持久化的，Redis 重启后，内容还在。

每个 Stream 都有唯一的名称，它就是 Redis 的 key，在我们首次使用 `xadd` 指令追加消息时自动创建。

每个 Stream 都可以挂多个消费组，每个消费组会有个游标 `last_delivered_id` 在 Stream 数组之上往前移动，表示当前消费组已经消费到哪条消息了。每个消费组都有一个 Stream 内唯一的名称，消费组不会自动创建，它需要单独的指令 `xgroup create` 进行创建，需要指定从 Stream 的某个消息 ID 开始消费，这个 ID 用来初始化 `last_delivered_id` 变量。

每个消费组 (Consumer Group) 的状态都是独立的，相互不受影响。也就是说同一份 Stream 内部的消息会被每个消费组都消费到。

同一个消费组 (Consumer Group) 可以挂接多个消费者 (Consumer)，这些消费者之间是竞争关系，任意一个消费者读取了消息都会使游标 `last_delivered_id` 往前移动。每个消费者有一个组内唯一名称。

消费者 (Consumer) 内部会有个状态变量 `pending_ids` ，它记录了当前已经被客户端读取的消息，但是还没有 ack。如果客户端没有 ack，这个变量里面的消息 ID 会越来越多，一旦某个消息被 ack，它就开始减少。这个 pending_ids 变量在 Redis 官方被称之为 `PEL` ，也就是 `Pending Entries List` ，这是一个很核心的数据结构，它用来确保客户端至少消费了消息一次，而不会在网络传输的中途丢失了没处理。

消息 ID

消息 ID 的形式是 `timestampInMillis-sequence` ，例如 `1527846880572-5` ，它表示当前的消息在毫米时间戳 `1527846880572` 时产生，并且是该毫秒内产生的第 5 条消息。消息 ID 可以由服务器自动生成，也可以由客户端自己指定，但是形式必须是 `整数-整数` ，而且必须是后面加入的消息的 ID 要大于前面的消息 ID。

消息内容

消息内容就是键值对，形如 hash 结构的键值对，这没什么特别之处。

增删改查

1. `xadd` 追加消息
2. `xdel` 删除消息，这里的删除仅仅是设置了标志位，不影响消息总长度
3. `xrange` 获取消息列表，会自动过滤已经删除的消息
4. `xlen` 消息长度
5. `del` 删除 Stream

```
# * 号表示服务器自动生成 ID，后面顺序跟着一堆 key/value
#  名字叫 laoqian，年龄 30 岁
127.0.0.1:6379> xadd codehole * name laoqian age 30
1527849609889-0  # 生成的消息 ID
127.0.0.1:6379> xadd codehole * name xiaoyu age 29
1527849629172-0
127.0.0.1:6379> xadd codehole * name xiaoqian age 1
1527849637634-0
127.0.0.1:6379> xlen codehole
(integer) 3
# -表示最小值 , + 表示最大值
127.0.0.1:6379> xrange codehole - +
127.0.0.1:6379> xrange codehole - +
1) 1) 1527849609889-0
   2) 1) "name"
      2) "laoqian"
      3) "age"
      4) "30"
2) 1) 1527849629172-0
   2) 1) "name"
      2) "xiaoyu"
      3) "age"
      4) "29"
3) 1) 1527849637634-0
   2) 1) "name"
      2) "xiaoqian"
      3) "age"
      4) "1"
# 指定最小消息 ID 的列表
127.0.0.1:6379> xrange codehole 1527849629172-0 +
1) 1) 1527849629172-0
   2) 1) "name"
```

```
      2) "xiaoyu"
      3) "age"
      4) "29"
2) 1) 1527849637634-0
   2) 1) "name"
      2) "xiaoqian"
      3) "age"
      4) "1"
# 指定最大消息 ID 的列表
127.0.0.1:6379> xrange codehole - 1527849629172-0
1) 1) 1527849609889-0
   2) 1) "name"
      2) "laoqian"
      3) "age"
      4) "30"
2) 1) 1527849629172-0
   2) 1) "name"
      2) "xiaoyu"
      3) "age"
      4) "29"
127.0.0.1:6379> xdel codehole 1527849609889-0
(integer) 1
# 长度不受影响
127.0.0.1:6379> xlen codehole
(integer) 3
# 被删除的消息没了
127.0.0.1:6379> xrange codehole - +
1) 1) 1527849629172-0
   2) 1) "name"
      2) "xiaoyu"
      3) "age"
      4) "29"
2) 1) 1527849637634-0
   2) 1) "name"
      2) "xiaoqian"
      3) "age"
      4) "1"
# 删除整个 Stream
127.0.0.1:6379> del codehole
(integer) 1
```

独立消费

我们可以在不定义消费组的情况下进行 Stream 消息的独立消费，当 Stream 没有新消息时，甚至可以阻塞等待。Redis 设计了一个单独的消费指令 `xread`，可以将 Stream 当成普通的消息队列 (list) 来使用。使用 `xread` 时，我们可以完

全忽略消费组 (Consumer Group) 的存在，就好比 Stream 就是一个普通的列表 (list)。

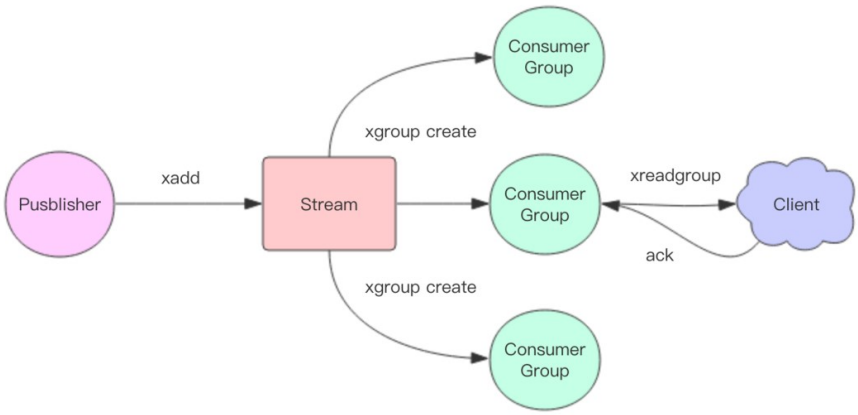
```
# 从 Stream 头部读取两条消息
127.0.0.1:6379> xread count 2 streams codehole 0-0
1) 1) "codehole"
   2) 1) 1) 1527851486781-0
         2) 1) "name"
            2) "laoqian"
            3) "age"
            4) "30"
      2) 1) 1527851493405-0
         2) 1) "name"
            2) "yurui"
            3) "age"
            4) "29"
# 从 Stream 尾部读取一条消息，毫无疑问，这里不会返回任何消息
127.0.0.1:6379> xread count 1 streams codehole $
(nil)
# 从尾部阻塞等待新消息到来，下面的指令会堵住，直到新消息到来
127.0.0.1:6379> xread block 0 count 1 streams codehole $
# 我们从新打开一个窗口，在这个窗口往 Stream 里塞消息
127.0.0.1:6379> xadd codehole * name youming age 60
1527852774092-0
# 再切换到前面的窗口，我们可以看到阻塞解除了，返回了新的消息内容
# 而且还显示了一个等待时间，这里我们等待了 93s
127.0.0.1:6379> xread block 0 count 1 streams codehole $
1) 1) "codehole"
   2) 1) 1) 1527852774092-0
         2) 1) "name"
            2) "youming"
            3) "age"
            4) "60"
(93.11s)
```

客户端如果想要使用 `xread` 进行顺序消费，一定要记住当前消费到哪里了，也就是返回的消息 ID。下次继续调用 `xread` 时，将上次返回的最后一个消息 ID 作为参数传递进去，就可以继续消费后续的消息。

block 0 表示永远阻塞，直到消息到来，block 1000 表示阻塞 1s，如果 1s 内没有任何消息到来，就返回 `nil` 。

```
127.0.0.1:6379> xread block 1000 count 1 streams codehole $
(nil)
(1.07s)
```

创建消费组



Stream 通过 `xgroup create` 指令创建消费组 (Consumer Group)，需要传递起始消息 ID 参数用来初始化 `last_delivered_id` 变量。

```
#  表示从头开始消费
127.0.0.1:6379> xgroup create codehole cg1 0-0
OK
# $ 表示从尾部开始消费，只接受新消息，当前 Stream 消息会全部忽略
127.0.0.1:6379> xgroup create codehole cg2 $
OK
# 获取 Stream 信息
127.0.0.1:6379> xinfo stream codehole
 1) length
 2) (integer) 3  # 共 3 个消息
 3) radix-tree-keys
 4) (integer) 1
 5) radix-tree-nodes
 6) (integer) 2
 7) groups
 8) (integer) 2  # 两个消费组
 9) first-entry  # 第一个消息
10) 1) 1527851486781-0
    2) 1) "name"
       2) "laoqian"
       3) "age"
       4) "30"
11) last-entry  # 最后一个消息
12) 1) 1527851498956-0
    2) 1) "name"
       2) "xiaoqian"
```

```
        3) "age"
        4) "1"
# 获取 Stream 的消费组信息
127.0.0.1:6379> xinfo groups codehole
1) 1) name
   2) "cg1"
   3) consumers
   4) (integer) 0  # 该消费组还没有消费者
   5) pending
   6) (integer) 0  # 该消费组没有正在处理的消息
2) 1) name
   2) "cg2"
   3) consumers  # 该消费组还没有消费者
   4) (integer) 0
   5) pending
   6) (integer) 0  # 该消费组没有正在处理的消息
```

消费

Stream 提供了 `xreadgroup` 指令可以进行消费组的组内消费，需要提供消费组名称、消费者名称和起始消息 ID。它同 `xread` 一样，也可以阻塞等待新消息。读到新消息后，对应的消息 ID 就会进入消费者的 PEL(正在处理的消息) 结构里，客户端处理完毕后使用 `xack` 指令通知服务器，本条消息已经处理完毕，该消息 ID 就会从 PEL 中移除。

```
# > 号表示从当前消费组的 last_delivered_id 后面开始读
# 每当消费者读取一条消息，last_delivered_id 变量就会前进
127.0.0.1:6379> xreadgroup GROUP cg1 c1 count 1 streams codehole >
1) 1) "codehole"
   2) 1) 1) 1527851486781-0
         2) 1) "name"
            2) "laoqian"
            3) "age"
            4) "30"
127.0.0.1:6379> xreadgroup GROUP cg1 c1 count 1 streams codehole >
1) 1) "codehole"
   2) 1) 1) 1527851493405-0
         2) 1) "name"
            2) "yurui"
            3) "age"
            4) "29"
127.0.0.1:6379> xreadgroup GROUP cg1 c1 count 2 streams codehole >
1) 1) "codehole"
   2) 1) 1) 1527851498956-0
         2) 1) "name"
```

```
                2) "xiaoqian"
                3) "age"
                4) "1"
        2) 1) 1527852774092-0
              2) 1) "name"
                 2) "youming"
                 3) "age"
                 4) "60"
# 再继续读取，就没有新消息了
127.0.0.1:6379> xreadgroup GROUP cg1 c1 count 1 streams codehole >
(nil)
# 那就阻塞等待吧
127.0.0.1:6379> xreadgroup GROUP cg1 c1 block 0 count 1 streams codehole >
# 开启另一个窗口，往里塞消息
127.0.0.1:6379> xadd codehole * name lanying age 61
1527854062442-0
# 回到前一个窗口，发现阻塞解除，收到新消息了
127.0.0.1:6379> xreadgroup GROUP cg1 c1 block 0 count 1 streams codehole >
1) 1) "codehole"
   2) 1) 1) 1527854062442-0
         2) 1) "name"
            2) "lanying"
            3) "age"
            4) "61"
(36.54s)
# 观察消费组信息
127.0.0.1:6379> xinfo groups codehole
1) 1) name
   2) "cg1"
   3) consumers
   4) (integer) 1  # 一个消费者
   5) pending
   6) (integer) 5  # 共 5 条正在处理的信息还有没有 ack
2) 1) name
   2) "cg2"
   3) consumers
   4) (integer) 0  # 消费组 cg2 没有任何变化，因为前面我们一直在操纵 cg1
   5) pending
   6) (integer) 0
# 如果同一个消费组有多个消费者，我们可以通过 xinfo consumers 指令观察每个消费者的状态
127.0.0.1:6379> xinfo consumers codehole cg1  # 目前还有 1 个消费者
1) 1) name
   2) "c1"
   3) pending
   4) (integer) 5  # 共 5 条待处理消息
   5) idle
   6) (integer) 418715  # 空闲了多长时间 ms 没有读取消息了
# 接下来我们 ack 一条消息
127.0.0.1:6379> xack codehole cg1 1527851486781-0
(integer) 1
```

```
127.0.0.1:6379> xinfo consumers codehole cg1
1) 1) name
   2) "c1"
   3) pending
   4) (integer) 4  # 变成了 5 条
   5) idle
   6) (integer) 668504
# 下面 ack 所有消息
127.0.0.1:6379> xack codehole cg1 1527851493405-0 1527851498956-0 1527852774092-0 1527854062442-0
(integer) 4
127.0.0.1:6379> xinfo consumers codehole cg1
1) 1) name
   2) "c1"
   3) pending
   4) (integer) 0  # pel 空了
   5) idle
   6) (integer) 745505
```

Stream 消息太多怎么办?

读者很容易想到，要是消息积累太多，Stream 的链表岂不是很长，内容会不会爆掉? `xdel` 指令又不会删除消息，它只是给消息做了个标志位。

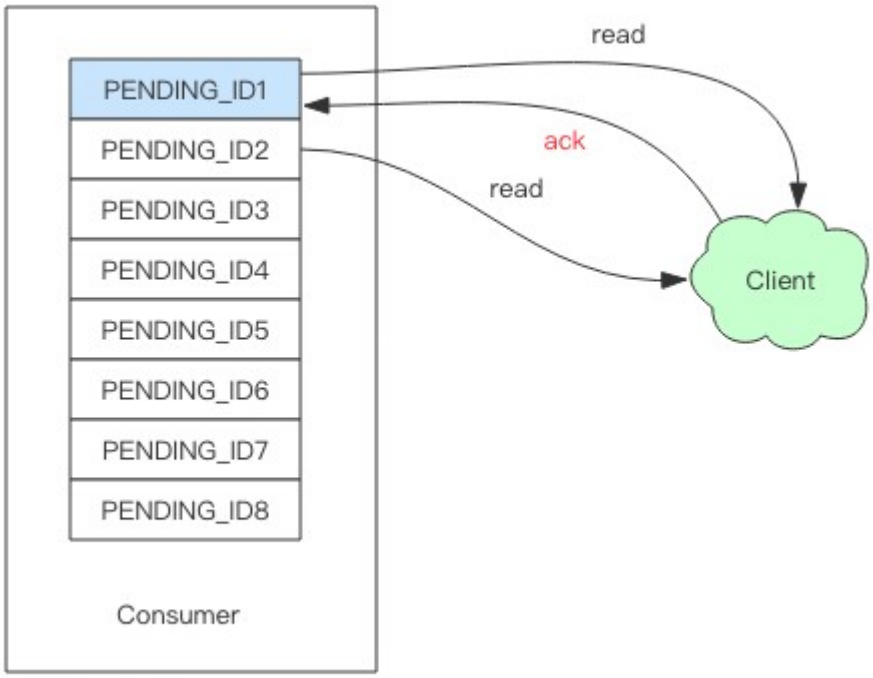
Redis 自然考虑到了这一点，所以它提供了一个定长 Stream 功能。在 `xadd` 的指令提供一个定长长度 `maxlen` ，就可以将老的消息干掉，确保最多不超过指定长度。

```
127.0.0.1:6379> xlen codehole
(integer) 5
127.0.0.1:6379> xadd codehole maxlen 3 * name xiaorui age 1
1527855160273-0
127.0.0.1:6379> xlen codehole
(integer) 3
```

我们看到 Stream 的长度被砍掉了。如果 Stream 在未来可以提供按时间戳清理消息的规则那就更加完美了，但是目前还没有。

消息如果忘记 ACK 会怎样?

Stream 在每个消费者结构中保存了正在处理中的消息 ID 列表 PEL，如果消费者收到了消息处理完了但是没有回复 ack，就会导致 PEL 列表不断增长，如果有很多消费组的话，那么这个 PEL 占用的内存就会放大。



PEL 如何避免消息丢失?

在客户端消费者读取 Stream 消息时，Redis 服务器将消息回复给客户端的过程中，客户端突然断开了连接，消息就丢失了。但是 PEL 里已经保存了发出去的消息 ID。待客户端重新连上之后，可以再次收到 PEL 中的消息 ID 列表。不过此时 `xreadgroup` 的起始消息 ID 不能为参数>，而必须是任意有效的消息 ID，一般将参数设为 0-0，表示读取所有的 PEL 消息以及自 `last_delivered_id` 之后的新消息。

Stream 的高可用

Stream 的高可用是建立主从复制基础上的，它和其它数据结构的复制机制没有区别，也就是说在 Sentinel 和 Cluster 集群环境下 Stream 是可以支持高可用的。不过鉴于 Redis 的指令复制是异步的，在 `failover` 发生时，Redis 可能会丢失极小部分数据，这点 Redis 的其它数据结构也是一样的。

分区 Partition

Redis 的服务器没有原生支持分区能力，如果想要使用分区，那就需要分配多个 Stream，然后在客户端使用一定的策略来生产消息到不同的 Stream。你也许会认为 Kafka 要先进很多，它是原生支持 Partition 的。关于这一点，我并不认同。记得 Kafka 的客户端也存在 HashStrategy 么，因为它也是通过客户端的 hash 算法来将不同的消息塞入不同分区的。

另外,Kafka 还支持动态增加分区数量的能力，但是这种调整能力也是很蹩脚的，它不会把之前已经存在的内容进行 rehash，不会重新分区历史数据。这种简单的动态调整的能力 Redis Stream 通过增加新的 Stream 就可以做到。

小结

Stream 的消费模型借鉴了 Kafka 的消费分组的概念，它弥补了 Redis Pub/Sub 不能持久化消息的缺陷。但是它又不同于 kafka，Kafka 的消息可以分 partition，而 Stream 不行。如果非要分 parition 的话，得在客户端做，提供不同的 Stream 名称，对消息进行 hash 取模来选择往哪个 Stream 里塞。

如果读者稍微研究过 Redis 作者的另一个开源项目 Disque 的话，这极可能是作者意识到 Disque 项目的活跃程度不够，所以将 Disque 的内容移植到了 Redis 里面。这只是本人的猜测，未必是作者的初衷。如果读者有什么不同的想法，可以在评论区一起参与讨论。