二

26 消息队列终极解决方案——Stream（上）

在 Redis 5.0 Stream 没出来之前，消息队列的实现方式都有着各自的缺陷，例如：

- 发布订阅模式 PubSub，不能持久化也就无法可靠的保存消息，并且对于离线重连的客户端不能读取历史消息的缺陷；
- 列表实现消息队列的方式不能重复消费，一个消息消费完就会被删除；
- 有序集合消息队列的实现方式不能存储相同 value 的消息，并且不能阻塞读取消息。

并且以上三种方式在实现消息队列时，只能存储单 value 值，也就是如果你要存储一个对象的情况下，必须先序列化成 JSON 字符串，在读取之后还要反序列化成对象才行，这也给用户的使用带来的不便，基于以上问题，Redis 5.0 便推出了 Stream 类型也是此版本最重要的功能，用于完美地实现消息队列，它借鉴了 Kafka 的设计思路，它支持消息的持久化和消息轨迹的消费，支持 ack 确认消息的模式，让消息队列更加的稳定和可靠。

接下来我们先来了解 Stream 自身的一些特性，然后在综合 Stream 的特性，结合 Java 代码完整的实现一个完美的消息队列示例。

基础使用

Stream 既然是一个数据类型，那么和其他数据类型相似，它也有一些自己的操作方法，例如：

- xadd 添加消息；
- xlen 查询消息长度；
- xdel 根据消息 ID 删除消息；
- del 删除整个 Stream；
- xrange 读取区间消息
- xread 读取某个消息之后的消息。

具体使用如下所述。

添加消息

```
127.0.0.1:6379> xadd key * name redis age 10
"1580880750844-0" #结果返回的是消息 id
```

其中 * 表示使用 Redis 的规则：时间戳 + 序号的方式自动生成 ID，用户也可以自己指定 ID。

相关语法：

```
xadd key ID field string [field string ...]
```

查询消息的长度

```
127.0.0.1:6379> xlen key
(integer) 1
```

相关语法：

```
xlen key
```

删除消息

```
127.0.0.1:6379> xadd key * name redis
"1580881585129-0" #消息 ID
127.0.0.1:6379> xlen key
(integer) 1
127.0.0.1:6379> xdel key 1580881585129-0 #删除消息，根据 ID
(integer) 1
127.0.0.1:6379> xlen key
(integer) 0
```

相关语法：

```
xdel key ID [ID ...]
```

此命令支持删除一条或多条消息，根据消息 ID。

删除整个 Stream

```
127.0.0.1:6379> del key #删除整个 Stream
(integer) 1
127.0.0.1:6379> xlen key
(integer) 0
```

相关语法：

```
del key [key ...]
```

此命令支持删除一个或多个 Stream。

查询区间消息

```
127.0.0.1:6379> xrange mq - +
1) 1) "1580882060464-0"
   2) 1) "name"
      2) "redis"
      3) "age"
      4) "10"
2) 1) "1580882071524-0"
   2) 1) "name"
      2) "java"
      3) "age"
      4) "20"
```

其中： - 表示第一条消息， + 表示最后一条消息。

相关语法：

```
xrange key start end [COUNT count]
```

查询某个消息之后的消息

```
127.0.0.1:6379> xread count 1 streams mq 1580882060464-0
1) 1) "mq"
   2) 1) 1) "1580882071524-0"
```

```
         2) 1) "name"
            2) "java"
            3) "age"
            4) "20"
```

在名称为 mq 的 Stream 中，从消息 ID 为 1580882060464-0 的，往后查询一条消息。

相关语法：

```
xread [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
```

此命令提供了阻塞读的参数 block，我们可以使用它读取从当前数据以后新增数据，命令如下：

```
127.0.0.1:6379> xread count 1 block 0 streams mq $
```

其中 `block 0` 表示一直阻塞，`$` 表示从最后开始读取，这个时候新开一个命令行插入一条数据，此命令展示的结果如下：

```
127.0.0.1:6379> xadd mq * name sql age 20 #新窗口添加数据
"1580890737890-0"
#阻塞读取到的新数据
127.0.0.1:6379> xread count 1 block 0 streams mq $
1) 1) "mq"
   2) 1) 1) "1580890737890-0"
         2) 1) "name"
            2) "sql"
            3) "age"
            4) "20"
(36.37s)
```

基础版消息队列

使用 Stream 消费分组实现消息队列的功能和列表方式的消息队列比较相似，使用 xadd 命令和 xread 循环读取就可以实现基础版的消息队列，具体代码如下：

```
import com.google.gson.Gson;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.StreamEntry;
import redis.clients.jedis.StreamEntryID;
import java.util.AbstractMap;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamExample {
    public static void main(String[] args) throws InterruptedException {
        // 消费者
        new Thread(() -> consumer()).start();
        Thread.sleep(1000);
        // 生产者
        producer();
    }
    /**
     * 生产者
     */
    public static void producer() throws InterruptedException {
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        // 推送消息
        Map<String, String> map = new HashMap<>();
        map.put("name", "redis");
        map.put("age", "10");
        // 添加消息
        StreamEntryID id = jedis.xadd("mq", null, map);
        System.out.println("消息添加成功 ID: " + id);
    }
    /**
     * 消费者
     */
    public static void consumer() {
        Jedis jedis = new Jedis("127.0.0.1", 6379);
        // 消费消息
        while (true) {
            // 获取消息，new StreamEntryID().LAST_ENTRY 标识获取当前时间以后的新增消息
            Map.Entry<String, StreamEntryID> entry = new AbstractMap.SimpleImmutabl
                    new StreamEntryID().LAST_ENTRY);
            // 阻塞读取一条消息（最大阻塞时间120s）
            List<Map.Entry<String, List<StreamEntry>>> list = jedis.xread(1, 120 *
            if (list.size() == 1) {
                // 读取到消息
                System.out.println("读取到消息 ID: " + list.get(0).getValue().get(0)
                // 使用 Gson 来打印 JSON 格式的消息内容
                System.out.println("内容: " + new Gson().toJson(list.get(0).getValue
            }
        }
    }
}
```

以上代码运行结果如下：

```
消息添加成功 ID：1580895735148-0
读取到消息 ID：1580895735148-0
内容：{"name":"redis","age":"10"}
```

以上代码需要特殊说明的是，我们使用 `new StreamEntryID().LAST_ENTRY` 来实现读取当前时间以后新增的消息，如果要从头读取历史消息把这行代码中的 `.LAST_ENTRY` 去掉即可。

还有一点需要注意，在 Jedis 框架中如果使用 jedis.xread() 方法来阻塞读取消息队列，第二个参数 long block 必须设置大于 0，如果设置小于 0，此阻塞条件就无效了，我查看了 jedis 的源码发现，它只有判断在大于 0 的时候才会设置阻塞属性，源码如下：

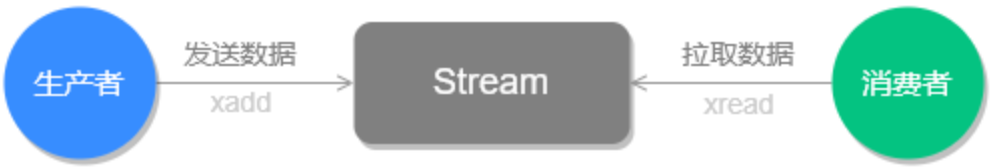
```
if (block > 0L) {
    params[streamsIndex++] = Keyword.BLOCK.raw;
    params[streamsIndex++] = Protocol.toByteArray(block);
}
```

所以 block 属性我们可以设置一个比较大的值来阻塞读取消息。

> 所谓的阻塞读取消息指的是当队列中没有数据时会进入休眠模式，等有数据之后才会唤醒继续执行。

小结

本文介绍了 Stream 的基础方法，并使用 xadd 存入消息和 xread 循环阻塞读取消息的方式实现了简易版的消息队列，交互流程如下图所示：



然后这些并不是 Stream 最核心的功能，下文我们将带领读者朋友们，使用消费分组来实现一个完美的消息队列。

上一页 下一页