# 链表

链表提供了高效的节点重排能力，以及顺序性的节点访问方式，并且可以通过增删节点来灵活地调整链表的长度。

作为一种常用数据结构，链表内置在很多高级的编程语言里面，因为Redis 使用的C 语言并没有内置这种数据结构，所以Redis 构建了自己的链表实现。

链表在Redis 中的应用非常广泛，比如列表键的底层实现之一就是链表。当一个列表键包含了数量比较多的元素，又或者列表中包含的元素都是比较长的字符串时， Redis 就会使用链表作为列表键的底层实现。

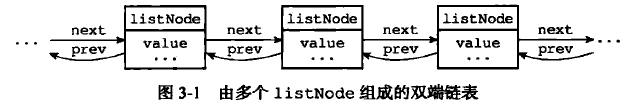
除了链表键之外，发布与订阅、慢查询、监视器等功能也用到了链表， Redis 服务器本身还使用链表来保存多个客户端的状态信息，以及使用链表来构建客户端输出缓冲区( output buffer ）。

## 链表和链表节点的实现

每个链表节点使用一个adlist.h/listNode 结构来表示：

|  |
| --- |
| typedef struct listNode {  // 前置节点  struct listNode •prev;  // 后置节点  struct listNode •next;  // 节点的值  void •value;  )listNode; |

多个listNode 可以通过prev 和next 指针组成双端链表，如图3-1 所示。



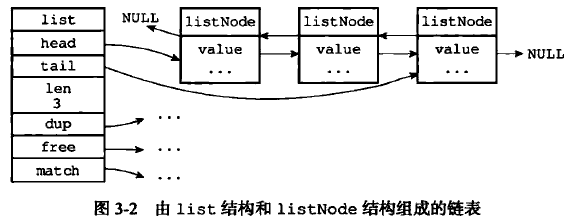
虽然仅仅使用多个listNode结构就可以组成链表，但使用adlist. h/ list 来持有链表的话，操作起来会更方便：

|  |
| --- |
| typedef struct list {  // 表头节点  listNode \*head;  // 表尾节点  listNode \*tail;  // 链表所包含的节点数量  unsigned long len;  // 节点值复制函数  void \* (\*dup) (void \*Ptr);  // 节点值释放函数  void (\*free) (void \*ptr);  // 节点值对比函数  int (\*mnatch) (void \*ptr,void \*key);  } list; |

list 结构为链表提供了表头指针head、表尾指针tail ，以及链表长度计数器len,而dup 、free 和match 成员则是用于实现多态链表所需的类型特定函数：

* dup 函数用于复制链表节点所保存的值；
* free 函数用于释放链表节点所保存的值；
* match 函数则用于对比链表节点所保存的值和另一个输入值是否相等。

图3-2 是由一个list 结构和三个listNode结构组成的链表。



Redis 的链表实现的特性可以总结如下：

* 双端：链表节点带有prev 和next 指针，获取某个节点的前置节点和后置节点的复杂度都是O(1)。
* 无环：表头节点的prev 指针和表尾节点的next 指针都指向NULL ，对链表的访问以NULL 为终点。
* 带表头指针和表尾指针：通过list 结构的head 指针和tail 指针，程序获取链表的表头节点和表尾节点的复杂度为O(1)。
* 带链表长度计数器：程序使用list 结构的len 属性来对list 持有的链表节点进行计数，程序获取链表中节点数量的复杂度为O(1)。
* 多态：链表节点使用void\*指针来保存节点值，并且可以通过list 结构的dup、free、match 三个属性为节点值设置类型特定函数，所以链表可以用于保存各种不同类型的值。

## 链表和链表节点的API

表3-1 列出了所有用于操作链表和链表节点的API。

