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，因为Redis使用的C语言并没有内置这种数据结构，所以Redis自己实现了链表。

链表在Redis中的应用非常广泛，比如列表的底层实现之一就是链表。当一个列表中包含的元素比较多时，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的底层实现。

除了列表之外，Redis中的发布与订阅、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。

在adlist.h中，链表节点的定义如下：

**typedef** struct listNode

**{**

struct listNode **\***prev**;**

struct listNode **\***next**;**

void **\***value**;**

**}** listNode**;**

多个listNode可以通过prev和next指针组成双向链表，如下图所示：

digraph {

    label = "\n 图 3-1    由多个 listNode 组成的双端链表"

    rankdir = LR;

    node [shape = record];

    //

    more_prev [label = "...", shape = plaintext];
    x [label = "<head> listNode | value \n ..."];
    y [label = "<head> listNode | value \n ..."];
    z [label = "<head> listNode | value \n ..."];
    more_next [label = "...", shape = plaintext];

    //

    more_prev -> x [label = "next"];
    x -> more_prev [label = "prev"];


    x -> y [label = "next"];
    y -> x [label = "prev"];

    y -> z [label = "next"];
    z -> y [label = "prev"];

    z -> more_next [label = "next"];
    more_next -> z [label = "prev"];
}

在adlist.h中，还定义了链表结构：

**typedef** struct list

**{**

listNode **\***head**;**

listNode **\***tail**;**

void **\*(\***dup**)(**void **\***ptr**);**

void **(\***free**)(**void **\***ptr**);**

int **(\***match**)(**void **\***ptr**,** void **\***key**);**

unsigned long len**;**

**}** list**;**

list结构为链表提供了头指针head、尾指针tail，以及链表长度计数器len，而函数指针dup，free和match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的值；

free函数用于释放链表节点的值；

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。

下图是由一个list结构和三个listNode结构组成的链表：

digraph {

    label = "\n 图 3-2    由 list 结构和 listNode 结构组成的链表"

    rankdir = LR;

    node [shape = record];

    //

    list [label = "list | <head> head | <tail> tail | <len> len \n 3 | <dup> dup | <free> free | <match> match ", width = 2.0];

    more_prev [label = "NULL", shape = plaintext];
    x [label = "<head> listNode | value \n ..."];
    y [label = "<head> listNode | value \n ..."];
    z [label = "<head> listNode | value \n ..."];
    more_next [label = "NULL", shape = plaintext];

    dup [label = "...", shape = plaintext];
    free [label = "...", shape = plaintext];
    match [label = "...", shape = plaintext];

    //

    list:head -> x;
    list:tail -> z;

    list:dup -> dup;
    list:free -> free;
    list:match -> match;

    x -> y;
    y -> x;

    y -> z;
    z -> y;

    //

    more_prev -> x [dir = back];
    z -> more_next;

}

Redis的链表实现的特性可以总结如下：

双向：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时间复杂度都是O(1)；

无环：表头节点的prey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；

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和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。

adlist.c是链表的实现源码文件，其中的代码都比较简单。比较有意思的是它内部实现了一个链表迭代器listIter，它的结构体定义如下：

**typedef** struct listIter **{**

listNode **\***next**;**

int direction**;**

**}** listIter**;**

其中，next表示使用迭代器当前指向的链表节点，对迭代器调用next操作，就返回该指针，并将next指向下一个节点。direction就表示迭代器的迭代方向，如果direction为AL\_START\_HEAD，表示迭代器从head开始从左到右迭代；如果direction为AL\_START\_TAIL，则表示迭代器从tail开始从右到左迭代。

迭代器的主要代码如下：

listIter **\***listGetIterator**(**list **\***list**,** int direction**)**

**{**

listIter **\***iter**;**

**if** **((**iter **=** zmalloc**(sizeof(\***iter**)))** **==** **NULL)** **return** **NULL;**

**if** **(**direction **==** AL\_START\_HEAD**)**

iter**->**next **=** list**->**head**;**

**else**

iter**->**next **=** list**->**tail**;**

iter**->**direction **=** direction**;**

**return** iter**;**

**}**

void listReleaseIterator**(**listIter **\***iter**)**

**{**

zfree**(**iter**);**

**}**

void listRewind**(**list **\***list**,** listIter **\***li**)**

**{**

li**->**next **=** list**->**head**;**

li**->**direction **=** AL\_START\_HEAD**;**

**}**

void listRewindTail**(**list **\***list**,** listIter **\***li**)**

**{**

li**->**next **=** list**->**tail**;**

li**->**direction **=** AL\_START\_TAIL**;**

**}**

listNode **\***listNext**(**listIter **\***iter**)**

**{**

listNode **\***current **=** iter**->**next**;**

**if** **(**current **!=** **NULL)** **{**

**if** **(**iter**->**direction **==** AL\_START\_HEAD**)**

iter**->**next **=** current**->**next**;**

**else**

iter**->**next **=** current**->**prev**;**

**}**

**return** current**;**

**}**

下面是一个使用迭代器进行链表复制的函数listDup：

list **\***listDup**(**list **\***orig**)**

**{**

list **\***copy**;**

listIter **\***iter**;**

listNode **\***node**;**

**if** **((**copy **=** listCreate**())** **==** **NULL)**

**return** **NULL;**

copy**->**dup **=** orig**->**dup**;**

copy**->**free **=** orig**->**free**;**

copy**->**match **=** orig**->**match**;**

iter **=** listGetIterator**(**orig**,** AL\_START\_HEAD**);**

**while((**node **=** listNext**(**iter**))** **!=** **NULL)** **{**

void **\***value**;**

**if** **(**copy**->**dup**)** **{**

value **=** copy**->**dup**(**node**->**value**);**

**if** **(**value **==** **NULL)** **{**

listRelease**(**copy**);**

listReleaseIterator**(**iter**);**

**return** **NULL;**

**}**

**}** **else**

value **=** node**->**value**;**

**if** **(**listAddNodeTail**(**copy**,** value**)** **==** **NULL)** **{**

listRelease**(**copy**);**

listReleaseIterator**(**iter**);**

**return** **NULL;**

**}**

**}**

listReleaseIterator**(**iter**);**

**return** copy**;**

**}**

其他关于redis的list代码，可以参考：

https://github.com/gqtc/redis-3.0.5/blob/master/redis-3.0.5/src/adlist.c