author：编程界的小学生

date：2021/03/18

一、底层数据结构

数据类型	底层数据结构
String	int/embstr/raw
List	ziplist/linkedlist/quicklist
Hash	ziplist/hashtable
Set	intset/hashtable
Zset	ziplist/skiplist

二、必知必会的铺垫

Redis对象，干嘛的？想象成对象头，不管你什么类型，都必须要带的，里面包含数据类型等等信息。

```
/*
 * Redis 对象
 */
typedef struct redisObject {
    // 类型 4bits，即上面[String,List,Hash,Set,Zset]中的一个
    unsigned type:4;
```

```
    // 编码方式 4bits，encoding表示对象底层所使用的编码。
    unsigned encoding:4;
    // LRU 时间（相对于 server.lruclock） 24bits
    unsigned lru:22;
    // 引用计数 Redis里面的数据可以通过引用计数进行共享 32bits
    int refcount;
    // 指向对象的值 64-bit
    void *ptr;
} robj;// 16bytes
```

一个RedisObject占用的字节数：4+4+24+32+64=128位/8=16字节。

三、sds/int/embstr/raw

1、SDS

1.1、sds概念

sds（simple dynamic string）：简单动态字符串。SDS只是字符串类型中存储字符串内容的结构，Redis中的字符串分为两种存储方式，分别是embstr和raw。

sds中包含了free(当前可用空间大小)，len(当前存储字符串长度)，buf[] (存储的字符串内容)，来看下SDS的源码：

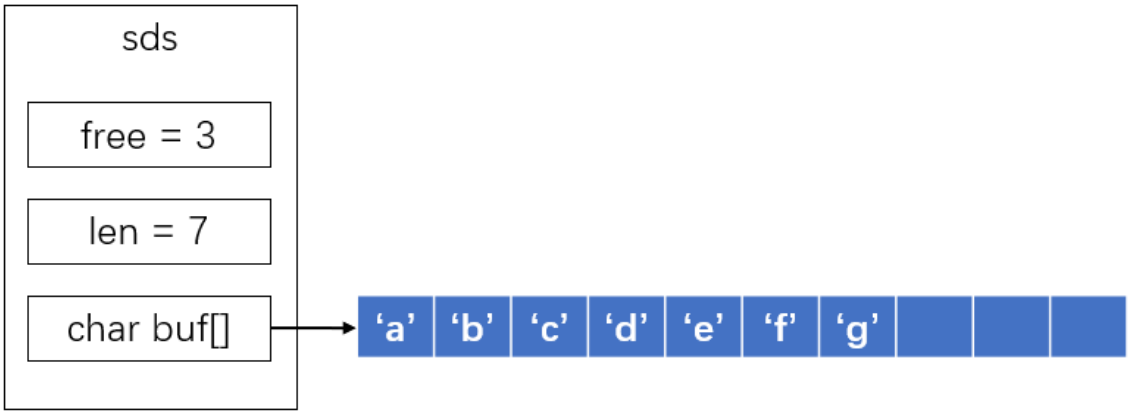
```
struct sdshdr{
    //记录buf数组中已使用字节的数量
    //等于 SDS 保存字符串的长度 4byte
    int len;
    //记录 buf 数组中未使用字节的数量 4byte
    int free;
    //字节数组，用于保存字符串 字节\0结尾的字符串占用了1byte
    char buf[];
}
```

> 包含了len、free、buf[]三个属性。那么他占用字节最少是：4+4+1=9字节。（仅限redis3.2版本之前。Redis3.2版本之后的sds结构发生了变化。）
>
> **先剧透一个知识点：字符串长度如果小于39的话，则采取embstr存储，否则采取raw类型存储。**
>
> 为啥是39？原因：对象头占16字节，空的sdshdr占用9字节，也就是一个数据至少占用16+9=25字节
>
> 其次操作系统使用jmalloc和tmalloc进行内存的分配，而内存分配的单位都是2的N次方，所以是2,4,8,16,32,64等字节，但是redis如果采取32的话，那么32-25=7，也太他妈少了，所以Redis采取的是64字节，所以：64-25=39。

1.2、sds干啥的?

比如你`set abc abcdefg`，简单的一个set会创建出两个sds，一个存key：abc，一个存value：abcdefg。

比如如下



1.3、为什么要有sds?

带着问题看答案：C语言中也有字符串类型，为啥她不用C的，反正他都是C语言写的，为啥要造个轮子sds?

1.3.1、优化获取字符串长度

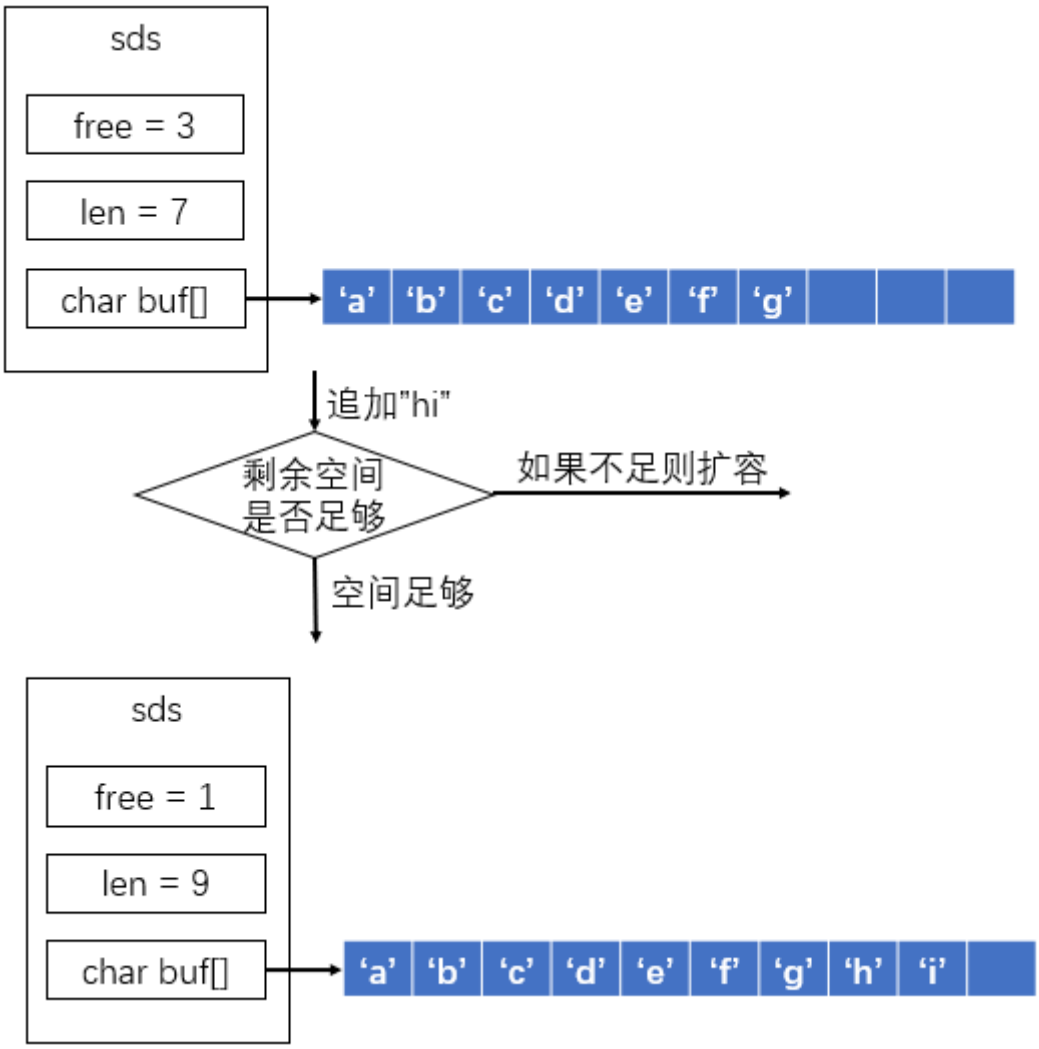
C语言要想获取字符串长度必须遍历整个字符串的每一个字符，然后自增做累加，时间复杂度为O(n)；sds直接维护了一个len变量，时间复杂度为O(1)。

1.3.2、减少内存分配

当我们对一个字符串类型进行追加的时候，可能会发生两种情况：

- 当前剩余空间(free)足够容纳追加内容时，我们就不需要再去分配内存空间，这样可以减少内存分配次数。
- 当前剩余空间不足以容纳追加内容，我们需要重新为其申请内存空间。

比如下面的sds的方式，free还有三个空余空间呢，你插入的是hi两个字符，所以足够，不需要调用函数重新分配，提升效率。



而C语言字符串在进行字符串的扩充和收缩的时候，都会面临着内存空间的重新分配问题。如果忘记分配或者分配大小不合理还会造成数据污染问题。

那么sds的free值哪来的呢？也就是字符串扩容策略

- 当给sds的值追加一个字符串，而当前的剩余空间不够时，就会触发sds的扩容机制。扩容采用了空间预分配的优化策略，即分配空间的时候：如果sds 值大小< 1M ,则增加一倍； 反之如果>1M , 则当前空间加1M作为新的空间。
- 当sds的字符串缩短了，sds的buf内会多出来一些空间，这个空间并不会马上被回收，而是暂时留着以防再用的时候进行多余的内存分配。这个是惰性空间释放的策略

1.3.3、惰性释放空间

当我们截断字符串时，Redis会把截断部分置空，只保留剩余部分，且不立即释放截断部分的内存空间，这样做的好处就是当下次再对这个字符串追加内容的时候，如果当前剩余空间足以容纳追加内容时，就不需要再去重新申请空间，避免了频繁的内存申请。暂时用不上的空间可以被Redis定时删除或者惰性删除。

1.3.4、防止缓冲区溢出

其实和减少内存分配是成套的，都是因为sds预先检查内存自动分配来做到防止缓冲区溢出的。比如：

程序中有两个在内存中紧邻着的 字符串 s1 和 s2，其中s1 保存了字符串“redis”，二s2 则保存了字符串“MongoDb”：

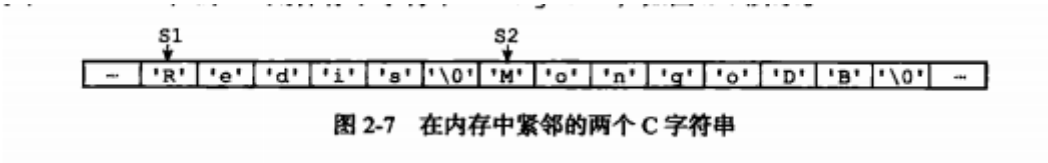


图 2-7　在内存中紧邻的两个 C 字符串

如果我们现在将s1 的内容修改为**redis cluster**，但是又忘了重新为s1 分配足够的空间，这时候就会出现以下问题：

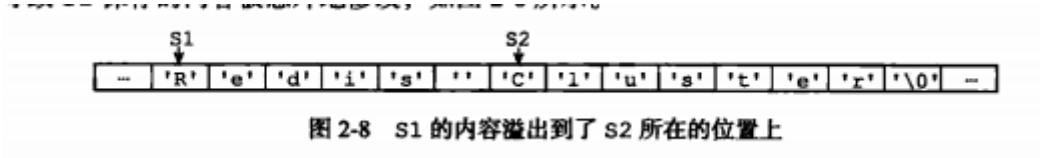


图 2-8　S1 的内容溢出到了 S2 所在的位置上

我们可以看到，原本s2 中的内容已经被S1的内容给占领了，s2 现在为 cluster，而不是“Mongodb”。造成了缓冲区溢出，也是数据污染。

Redis中SDS的空间分配策略完全杜绝了发生缓冲区溢出的可能性：

当我们需要对一个SDS 进行修改的时候，redis 会在执行拼接操作之前，预先检查给定SDS 空间是否足够，如果不够，会先拓展SDS 的空间，然后再执行拼接操作

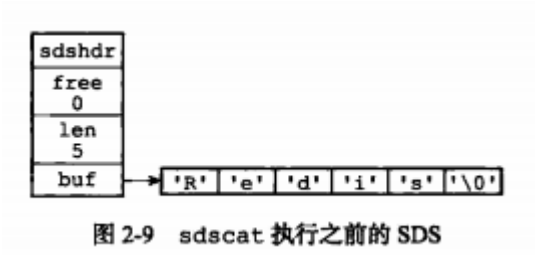


图 2-9　sdscat 执行之前的 SDS

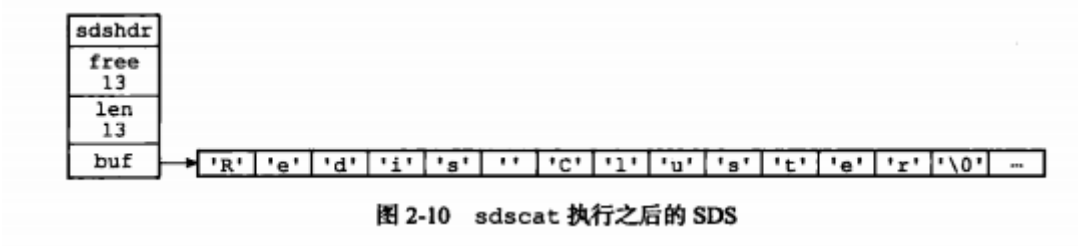
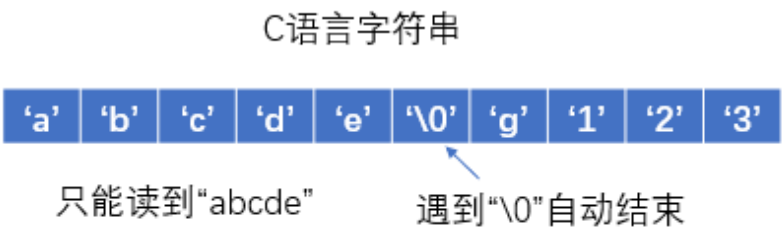
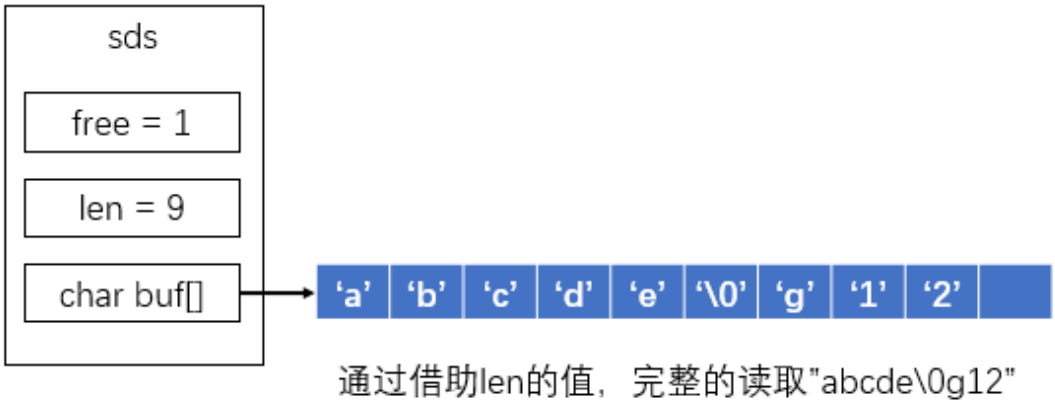


图 2-10　sdscat 执行之后的 SDS

1.3.5、二进制安全

在C语言中通过判断当前字符是否为'\0'来确定字符串是否结束，而在sds结构中，只要遍历长度没有达到len，即使遇到'\0'，也不会认为字符串结束。比如下面内存，C语言的字符串类型会丢失g123这四个字符，因为他遇到'\0'就结束了，而sds不会存在此问题。





1.3.6、与C总结

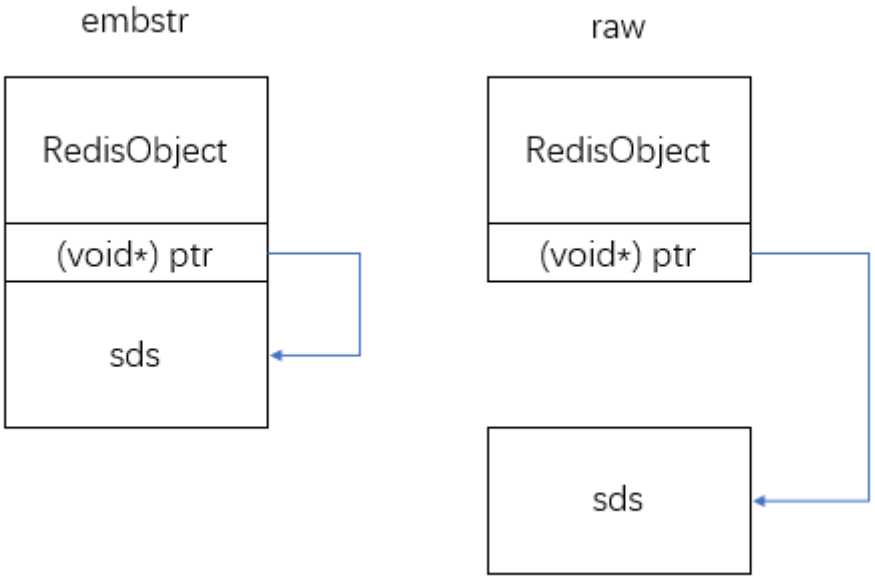
C 字符串	SDS
获取字符串长度的复杂度为O（N）	获取字符串长度的复杂度为O(1)
API 是不安全的，可能会造成缓冲区溢出	API 是安全的，不会造成缓冲区溢出
修改字符串长度N次**必然需要**执行N次内存重分配	修改字符串长度N次**最多执行**N次内存重分配
只能保存文本数据，二进制不安全	可以保存二进制数据和文本文数据，二进制安全

2、int

如果一个字符串内容可转为 long，那么该字符串会被转化为 long 类型，redisObject的对象 ptr 指向该 long，并将 encoding 设置为 int，这样就不需要重新开辟空间，算是长整形的一个优化。

3、embstr/raw

上面的SDS只是字符串类型中存储字符串内容的结构，Redis中的字符串分为两种存储方式，分别是embstr和raw，当字符串长度特别短（redis3.2之前是39字节，redis3.2之后是44字节）的时候，Redis使用embstr来存储字符串，而当字符串长度超过39（redis3.2之前）的时候，就需要用raw来存储，下面是他们的字符串完整结构的示意图：



embstr的存储方式是将RedisObject对象头和SDS结构放在内存中连续的空间位置，也就是使用malloc方法一次分配，而raw需要两次malloc，分别分配对象头和SDS的空间。释放空间也一样，embstr释放一次，raw释放两次，所以embstr是一种优化，但是为什么是39字节才采取embstr呢？39哪来的？

这个问题在上面sds里已经说过了。

原因：对象头占16字节，空的sdshdr占用9字节，也就是一个数据至少占用16+9=25字节。

其次操作系统使用jmalloc和tmalloc进行内存的分配，而内存分配的单位都是2的N次方，所以是2,4,8,16,32,64等字节，但是redis如果采取32的话，那么32-25=7，也太他妈少了，所以Redis采取的是64字节，所以：64-25=39。

（仅限redis3.2版本之前。Redis3.2版本之后的sds结构发生了变化【最小的是sdshdr5，空的话占用3字节+1个空白=4字节，16+4=20；64-20=44】。）

4、总结

1. redis的string底层数据结构使用的是sds，但是sds有两种存储方式，一种是embstr，一种是raw。

2. embstr的优势在于和对象头一起分配到连续空间，只需要调用函数malloc一次就行。raw需要两次，一次是对象头，一次是sds。释放也一样，embstr释放一次，raw释放两次。

3. 字符串内容可转为 long，采用 int 类型，否则长度<39（3.2版本前是39，3.2版本后分界线是44） 用 embstr，其他用 raw。

4. SDS 是Redis自己构建的一种简单动态字符串的抽象类型，并将 SDS 作为 Redis 的默认字符串表示。

5. SDS 与 C 语言字符串结构相比，具有四大优势。

四、ziplist/linkedlist/quicklist

1、ziplist

2、linkedlist

3、quicklist

五、hashtable

六、intset

七、skiplist