Redis 设计与实现之 SDS -- 阅读笔记

作者：guoweikuang
原文链接：https://ld246.com/article/1540484060554
来源网站：链滴

一、简单动态字符串（SDS）

简单动态字符串（simple dynamic string, SDS) 是 Redis 实现的一个用于保存字符串的数据结构，Redis 没有使用C 语言传统的字符串表示。

比如：

```
redis> set msg "hello, world"
创建一个键值对
键值对的 键 是一个字符串对象，对象的底层实现是一个保存着字符串"msg"的SDS
键值对的 值 是一个字符串对象，对象的底层实现是一个保存着字符串"hello, world" 的 SDS
```

SDS 的定义

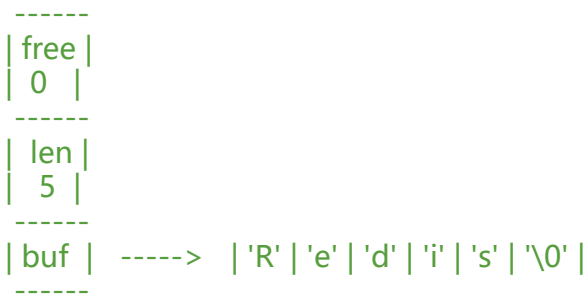
```
struct sdshdr {
 // 记录 buf 数组中已使用字节的数量
 // 等于 SDS 所保存字符串的长度
 int len;

 // 记录 buf 数组中未使用字节的数量
 int free;

 // 字节数组，用于保存字符串
 char buf[];
}
```

比如保存一个 "Redis" 字符串：

```
 ------
|sdshdr|
```



free 值为0，表示这个 SDS 没有分配任未使用空间，如果free > 0, 这个 SDS 为 buf 数组分配对应值未使用空间，比如 free = 5, 会在 buf 数组的 "\0" 后分配5字节未使用的空间
len 值为5，表示这个 SDS 保存了一个5字节长的字符串

SDS 和 C 字符串的区别

SDS 比 C 字符串更适用于 Redis 的原因?

1、C 字符串不记录自身的长度，导致获取长度需要遍历字符串， 复杂度为O(N)。而 SDS 结构的 len 属性记录了 SDS 本身的长度，复杂度为O(1)
2、当使用 char *strcat(chart *dest, const chart *src) 函数时，如果 dest 没有分配足够的内存容纳 src， 将产生缓冲区溢出的问题，导致溢出到相邻的字符的空间中，修改了相邻字符的内容
3、SDS 空间分配策略会检查 SDS 的空间是否满足所需的需求，然后通过 SDS API 自动将空间扩展所需的大小，就不会出现缓冲区溢出问题
4、减少修改字符串时带来的内存重分配次数，如果是增长字符串的操作，执行操作之前会通过内存分配来扩展底层数组的空间大小，如果是缩短字符串的操作，执行操作之后通过内存重分配来释放字串不再使用的部分空间。内存重分配涉及复杂的算法，是一个耗时操作

C 字符串 与 SDS 区别

C 字符串	SDS
获取字符串长度复杂度O(N)	获取字符串长度复杂O(1)
API 不安全，容易缓冲区溢出	API 安全，缓冲区溢出
修改字符串长度N次需要执行 N 次内存重分配	修字符串长度N次<= N次内存重分配
只能保持文本数据	保持文本及二进制数据
使用所有库函数	只能使用部分库函数

SDS 空间预分配和惰性空间释放策略

1、空间预分配

空间预分配用于优化 SDS 的字符串增长操作，其实就是当需要对 SDS 空间扩展时，除了分配所需空外，还会为 SDS 分配额外的未使用空间。额外分配的空间有两种情况：

1、修改 SDS 后, SDS 的长度(len属性) < 1MB, 将分配和 len 大小一样的未使用空间，len 的值和 free 值相同
(如修改后 SDS = 13 字节，程序分配了 13 字节未使用空间，SDS 的 buf 数组长度 = 13 + 13 + 1 = 7 字节)
2、修改 SDS 后，SDS 的长度 > 1MB, 将分配 1MB 的未使用空间
(如修改后 SDS = 20MB，SDS 的 buf 数组长度 = 20MB + 1MB + 1byte(保存空字符))

2、惰性空间释放

惰性空间释放用于优化 SDS 的字符串缩短操作，当 SDS API 需要缩短 SDS 保存的字符串长度时，立即使用内存重分配回收，而是使用 free 记录这些字节的数量。不释放多出来的字节空间，如果这候对 SDS 进行增长操作，这些未使用空间就可以使用。

总结

Redis 使用 C 字符串作为字面量，在大多数情况下，使用 SDS (简单动态字符串) 作为字符串表示，有对比了 C 字符串与 SDS 的区别，并指出 SDS 的优势等