

### 1、Redis字符串的实现

Redis虽然是用C语言写的，但却没有直接用C语言的字符串，而是自己实现的一套字符串。目的就是为了提升速度，提升性能。

Redis构建了一个叫做简单动态字符串（Simple Dynamic String），简称SDS

#### 1.1 SDS代码结构

**struct** **sdshdr**{  
    *//  记录已使用长度*

**int** len;  
    *// 记录空闲未使用的长度*  
    **int** free;  
    *// 字符数组*  
    **char**[] buf;  
};

重点借鉴Redis的设计思路：



Redis的字符串也会遵循C语言的字符串的实现规则，即最后一个字符为空字符。然而这个空字符不会被计算在len里头。

#### 1.2 SDS的动态扩展特点

动态变化长度



如上图所示刚开始s1只有5个空闲位子，后面需要追加' world'6个字符，很明显是不够的。那咋办？Redis会做以下三个操作：

1. 计算出大小是否足够
2. 开辟空间至满足所需大小
3. 开辟与已使用大小len相同长度的空闲free空间（如果len < 1M），开辟1M长度的空闲free空间（如果len >= 1M）

看到这儿为止有没有觉得这个实现跟Java的列表List实现有点类似呢？

### 2、Redis字符串的性能优势

* 快速获取字符串长度
* 避免缓冲区溢出
* 降低空间分配次数提升内存使用效率

#### 2.1 快速获取字符串长度

由于SDS结构体里以及存了已使用字符长度len，所以当想获取字符串长度时直接返回len即可，时间复杂度为O(1)。如果使用C语言的字符串的话它的字符串长度获取函数时间复杂度为O(n)，n为字符个数，因为他从头到尾遍历（到空字符’\0’）相加。

#### 2.2 避免缓冲区溢出

对一个C语言字符串进行strcat追加字符串的时候需要提前开辟需要的空间，如果不开辟空间的话可能会造成缓冲区溢出，而影响程序其他代码。如下图，有一个字符串s1="hello" 和 字符串s2="baby",现在要执行strcat(s1,"world"),并且执行前未给s1开辟空间，所以造成了缓冲区溢出。



而对于Redis而言由于每次追加字符串时都会检查空间是否够用，所以不会存在缓冲区溢出问题。每次追加操作前都会做如下操作：

1. 计算出大小是否足够
2. 开辟空间至少满足所需大小

#### 2.3 降低空间分配次数提升内存使用效率

字符串的追加操作会涉及到内存分配问题，然而内存分配问题会牵扯内存划分算法以及系统调用所以如果频繁发生的话影响性能，所以对于性能至上的Redis来说这是万万不能忍受的。所以采取了以下两种优化措施：

* 空间预分配
* 惰性空间回收

空间预分配

对于追加操作来说，Redis不仅会开辟空间至够用而且还会预分配未使用的空

间(free)来用于下一次操作。至于未使用的空间(free)的大小则由修改后的字符串长度决定。

当修改后的字符串长度len < 1M,则会分配与len相同长度的未使用的空间(free)

当修改后的字符串长度len >= 1M,则会分配1M长度的未使用的空间(free)

有了这个预分配策略之后会减少内存分配次数，因为分配之前会检查已有的free空间是否够，如果够则不开辟了～

惰性空间回收

与上面情况相反，惰性空间回收适用于字符串缩减操作。比如有个字符串

s1="hello world"，对s1进行sdstrim(s1," world")操作，执行完该操作之后Redis不会立即回收减少的部分，而是会分配给下一个需要内存的程序。当然，Redis也提供了回收内存的api,可以自己手动调用来回收缩减部分的内存。