# 简单动态字符串

# 01 / 什么是简单动态字符串?

1. Simple Dynamic String，是Redis内部自定义的一种数据类型;
2. 在Redis数据库内部，包含字符串的键值对在底层都是由SDS实现的;
3. SDS还被用于缓冲区的实现，如AOF缓冲区、客户端中的输入缓冲区。

```
set text "hello world"
rpush names "john" "lucy" "tony"
sadd users "liubei" "guanyu" "zhangfei"
```

## 02 / 为什么不用C原生字符串?

1. C语言没有为字符串设计专门的数据类型，而是将字符串存储在char类型的数组中；
2. C语言用空字符（\0）标记字符串的结束，空字符不是数字0，它的ASCII码值为0。

| | |
|---|---|
| 获取长度的复杂度高 | C字符串不记录自身长度，程序必须遍历整个字符串统计其长度，复杂度为O(N)。 |
| 内存重分配十分频繁 | 几乎每次修改C字符串，程序就要对保存这个字符串的数组重新分配一次内存空间。 |
| 不能保证二进制安全 | 因为C字符串以空字符结尾，所以不适合保存二进制数据（内部可能携带空字符）。 |

```
/* sds.h */
struct sdshdr {
    unsigned int len;      // 已使用的字节数量
    unsigned int free;     // 未使用的字节数量
    char buf[];            // 保存字符串的数组
};
```

```
/* sds.h */
struct sdshdr {
    unsigned int len;      // 已使用的字节数量
    unsigned int free;     // 未使用的字节数量
    char buf[];            // 保存字符串的数组
};
```

1. 降低了获取字符串长度的复杂度

   SDS在len属性中记录了字符串的实际长度，所以获取长度的复杂度仅仅为O(1)；

2. 减少了修改带来的内存分配次数

   通过空间预分配和惰性空间释放策略，优化了修改字符串时所需的内存分配次数；

3. 保证了二进制数据存储的安全性

   SDS不会对buf中的数据做任何限制，因为它采用len属性来判定字符串是否结束；

   SDS依然以空字符结尾，这样其内部可以很方便的重用一部分C字符串库中的函数。

```
/* sds.h */
struct sdshdr {
    unsigned int len;      // 已使用的字节数量
    unsigned int free;     // 未使用的字节数量
    char buf[];            // 保存字符串的数组
};
```

| | |
|---|---|
| 预分配 | 用于优化增长操作，即不仅为其分配存放字符串所需的空间，还会为其分配额外的未使用空间；<br>若修改后SDS的长度小于1MB，则分配的未使用空间与len相同，否则分配的未使用空间为1MB； |
| 惰性释放 | 用于优化缩短操作，当缩短SDS时程序不立刻重新分配内存，而是使用free属性记录这些字节； |

```
/* sds.h */
struct sdshdr {
    unsigned int len;      // 已使用的字节数量
    unsigned int free;     // 未使用的字节数量
    char buf[];            // 保存字符串的数组
};
```

不足之处：

len、free统一占据4字节，对于较短的字符串来说，浪费了存储空间。

```
/* sds.h */
struct __attribute__ ((__packed__)) sdshdr5 {
    unsigned char flags;  // 低3位存储类型，高5位存储长度
    char buf[];           // 存放实际的内容
};
struct __attribute__ ((__packed__)) sdshdr8 {
    uint8_t len;          // 使用的字节数量
    uint8_t alloc;        // 全部的字节数量
    unsigned char flags;  // 低3位存储类型，高5位预留
    char buf[];           // 存放实际的内容
};
...
```

优化方向：

通过字符串长度，将其分为5种类型，分别为1字节、2字节、4字节、8字节、小于1字节。

```
/* sds.h */
struct __attribute__ ((__packed__)) sdshdr5 {
    unsigned char flags;  // 低3位存储类型，高5位存储长度
    char buf[];           // 存放实际的内容
};
struct __attribute__ ((__packed__)) sdshdr8 {
    uint8_t len;          // 使用的字节数量
    uint8_t alloc;        // 全部的字节数量
    unsigned char flags;  // 低3位存储类型，高5位预留
    char buf[];           // 存放实际的内容
};
...
```

结构体默认会按其所有变量大小的最小公倍数做字节对齐，
使用packed修饰后，则变为按照1字节对齐，可以进一步节约内存。

牛客大学
-专业求职辅导-

# THANKS

关注【牛客大学】公众号
回复“牛客大学”获取更多求职资料