# Redis 学习笔记

## Sds 简单动态字符串

typedef char \*sds; **//类型别名，用于指向 sdshdr 的 buf 属性**

**/\***

**\*保存字符串对象的结构**

**\*/**

struct sdshdr {

int len; **// buf 中已占用空间的长度**

int free; **// buf 中剩余可用空间的长度**

char buf[]; **// 数据空间**

};

**Sizeof(struct sdshdr) 长度为8。Char buf[]在结构体中的长度为0。**

**(void\*) (s-(sizeof(struct sdshdr)))获取结构体首地址。**

**Sds总结：**

1. **获取字符串长度和字符串空余长度的复杂度为O(1)**
2. **API是安全的，不会造成缓冲区溢出**
3. **修改字符串长度N次最多需要执行N次内存重分配**
4. **可以保存文本或者二进制数据**
5. **可以使用一部分<string.h>库中的函数**

### 函数

#### 1.sdsnewlen

**根据给定的init字符串和字符串长度initlen创建一个新的sds。**

**sds sdsnewlen(const void \*init, size\_t initlen) {**

**struct sdshdr \*sh;**

**// 根据是否有初始化内容，选择适当的内存分配方式**

**// T = O(N)**

**if (init) {**

**// zmalloc 不初始化所分配的内存**

**sh = zmalloc(sizeof(struct sdshdr)+initlen+1);**

**} else {**

**// zcalloc 将分配的内存全部初始化为 0**

**sh = zcalloc(sizeof(struct sdshdr)+initlen+1);**

**}**

**// 内存分配失败，返回**

**if (sh == NULL) return NULL;**

**// 设置初始化长度**

**sh->len = initlen;**

**// 新 sds 不预留任何空间**

**sh->free = 0;**

**// 如果有指定初始化内容，将它们复制到 sdshdr 的 buf 中**

**// T = O(N)**

**if (initlen && init)**

**memcpy(sh->buf, init, initlen);**

**// 以 \0 结尾**

**sh->buf[initlen] = '\0';**

**// 返回 buf 部分**

**return (char\*)sh->buf;**

**}**

#### Sdsempty

创建一个空字符串“ ”的sds

**sds sdsempty(void) {**

**return sdsnewlen("",0); //调用sdsnewlen进行创建**

**}**

#### Sdsnew

根据给定的字符串init，创建一个包含同样字符串的sds，本质是调用sdsnewlen。

**sds sdsnew(const char \*init) {**

**size\_t initlen = (init == NULL) ? 0 : strlen(init);//对init进行判断，如果为空，字符串长度为0**

**return sdsnewlen(init, initlen);**

**}**

#### Sdsdup

复制给定sds的副本，该函数参照strdup。调用的是sdsnewlen来复制sds.

**sds sdsdup(const sds s) {**

**return sdsnewlen(s, sdslen(s));**

**}**

#### Sdsfree

释放给定的sds。

**void sdsfree(sds s) {**

**if (s == NULL) return;**

**zfree(s-sizeof(struct sdshdr));**

**}**

#### Sdsclear

并不释放sds的字符串空间，而是重置sds所保持的字符串，只是简单的将len和free进行修改，并将buf的第一个字符置为’\0’。

**void sdsclear(sds s) {**

**// 取出 sdshdr**

**struct sdshdr \*sh = (void\*) (s-(sizeof(struct sdshdr)));**

**// 重新计算属性**

**sh->free += sh->len;**

**sh->len = 0;**

**// 将结束符放到最前面（相当于惰性地删除 buf 中的内容）**

**sh->buf[0] = '\0';**

**}**

#### sdsMakeRoomFor

对sds中buf的长度进行增加，确保函数执行之后，buf至少会有addlen+1的空余空间。在对buf的长度进行扩展的过程中，如果新长度超过1MB，那么新长度分配的空间将会是newlen + 1MB，反之则是newlen \* 2。

**sds sdsMakeRoomFor(sds s, size\_t addlen) {**

**struct sdshdr \*sh, \*newsh;**

**// 获取 s 目前的空余空间长度**

**size\_t free = sdsavail(s);**

**size\_t len, newlen;**

**// s 目前的空余空间已经足够，无须再进行扩展，直接返回**

**if (free >= addlen) return s;**

**// 获取 s 目前已占用空间的长度**

**len = sdslen(s);**

**sh = (void\*) (s-(sizeof(struct sdshdr)));**

**// s 最少需要的长度**

**newlen = (len+addlen);**

**// 根据新长度，为 s 分配新空间所需的大小**

**if (newlen < SDS\_MAX\_PREALLOC)**

**// 如果新长度小于 SDS\_MAX\_PREALLOC**

**// 那么为它分配两倍于所需长度的空间**

**newlen \*= 2;**

**else**

**// 否则，分配长度为目前长度加上 SDS\_MAX\_PREALLOC**

**newlen += SDS\_MAX\_PREALLOC;**

**// T = O(N)**

**newsh = zrealloc(sh, sizeof(struct sdshdr)+newlen+1);**

**// 内存不足，分配失败，返回**

**if (newsh == NULL) return NULL;**

**// 更新 sds 的空余长度**

**newsh->free = newlen - len;**

**// 返回 sds**

**return newsh->buf;**

**}**

#### sdsRemoveFreeSpace

回收sds中的空余空间，回收不会对sds中已保存的字符串内容做任何修改。本质是重新分配内存。

**sds sdsRemoveFreeSpace(sds s) {**

**struct sdshdr \*sh;**

**sh = (void\*) (s-(sizeof(struct sdshdr)));**

**// 进行内存重分配，让 buf 的长度仅仅足够保存字符串内容**

**// T = O(N)**

**sh = zrealloc(sh, sizeof(struct sdshdr)+sh->len+1);**

**// 空余空间为 0**

**sh->free = 0;**

**return sh->buf;**

**}**

#### sdsAllocSize

返回sds的内存字节数。Len+free+1。

**size\_t sdsAllocSize(sds s) {**

**struct sdshdr \*sh = (void\*) (s-(sizeof(struct sdshdr)));**

**return sizeof(\*sh)+sh->len+sh->free+1;**

**}**

#### sdsIncrLen

根据参数incr，增加sds长度，缩减空余空间。并将\0放到新字符串的尾端。

该函数是在调用 sdsMakeRoomFor() 对字符串进行扩展，然后用户在字符串尾部写入了某些内容之后，用来正确更新 free 和 len 属性的。(个人感觉该函数无用)

**void sdsIncrLen(sds s, int incr) {**

**struct sdshdr \*sh = (void\*) (s-(sizeof(struct sdshdr)));**

**// 确保 sds 空间足够**

**assert(sh->free >= incr);**

**// 更新属性**

**sh->len += incr;**

**sh->free -= incr;**

**// 这个 assert 其实可以忽略**

**// 因为前一个 assert 已经确保 sh->free - incr >= 0 了**

**assert(sh->free >= 0);**

**// 放置新的结尾符号**

**s[sh->len] = '\0';**

**}**

#### sdsgrowzero

将sds扩充至指定长度，未使用的部分以0字节填充。

**sds sdsgrowzero(sds s, size\_t len) {**

**struct sdshdr \*sh = (void\*)(s-(sizeof(struct sdshdr)));**

**size\_t totlen, curlen = sh->len;**

**// 如果 len 比字符串的现有长度小，**

**// 那么直接返回，不做动作**

**if (len <= curlen) return s;**

**// 扩展 sds**

**// T = O(N)**

**s = sdsMakeRoomFor(s,len-curlen);**

**// 如果内存不足，直接返回**

**if (s == NULL) return NULL;**

**/\* Make sure added region doesn't contain garbage \*/**

**// 将新分配的空间用 0 填充，防止出现垃圾内容**

**// T = O(N)**

**sh = (void\*)(s-(sizeof(struct sdshdr)));**

**memset(s+curlen,0,(len-curlen+1)); /\* also set trailing \0 byte \*/**

**// 更新属性**

**totlen = sh->len+sh->free;**

**sh->len = len;**

**sh->free = totlen-sh->len;**

**// 返回新的 sds**

**return s;**

**}**

#### Sdscatlen

将长度为len的字符串t追加到sds的字符串末尾。

**sds sdscatlen(sds s, const void \*t, size\_t len) {**

**struct sdshdr \*sh;**

**// 原有字符串长度**

**size\_t curlen = sdslen(s);**

**// 扩展 sds 空间**

**// T = O(N)**

**s = sdsMakeRoomFor(s,len);**

**// 内存不足？直接返回**

**if (s == NULL) return NULL;**

**// 复制 t 中的内容到字符串后部**

**// T = O(N)**

**sh = (void\*) (s-(sizeof(struct sdshdr)));**

**memcpy(s+curlen, t, len);**

**// 更新属性**

**sh->len = curlen+len;**

**sh->free = sh->free-len;**

**// 添加新结尾符号**

**s[curlen+len] = '\0';**

**// 返回新 sds**

**return s;**

**}**

#### sdscat

将给定字符串 t 追加到 sds 的末尾.

**sds sdscat(sds s, const char \*t) {**

**return sdscatlen(s, t, strlen(t));**

**}**

#### sdscatsds

将另一个 sds 追加到一个 sds 的末尾.

**sds sdscatsds(sds s, const sds t) {**

**return sdscatlen(s, t, sdslen(t));**

**}**

#### sdscpylen

将字符串 t 的前 len 个字符复制到 sds s 当中，并在字符串的最后添加终结符。

**sds sdscpylen(sds s, const char \*t, size\_t len) {**

**struct sdshdr \*sh = (void\*) (s-(sizeof(struct sdshdr)));**

**// sds 现有 buf 的长度**

**size\_t totlen = sh->free+sh->len;**

**// 如果 s 的 buf 长度不满足 len ，那么扩展它**

**if (totlen < len) {**

**// T = O(N)**

**s = sdsMakeRoomFor(s,len-sh->len);**

**if (s == NULL) return NULL;**

**sh = (void\*) (s-(sizeof(struct sdshdr)));**

**totlen = sh->free+sh->len;**

**}**

**// 复制内容**

**// T = O(N)**

**memcpy(s, t, len);**

**// 添加终结符号**

**s[len] = '\0';**

**// 更新属性**

**sh->len = len;**

**sh->free = totlen-len;**

**// 返回新的 sds**

**return s;**

**}**

#### sdscpy

将字符串复制到 sds 当中，覆盖原有的字符。

**sds sdscpy(sds s, const char \*t) {**

**return sdscpylen(s, t, strlen(t));**

**}**

#### sdsll2str

将long long 类型的整数转换成字符串。

**int sdsll2str(char \*s, long long value) {**

**char \*p, aux;**

**unsigned long long v;**

**size\_t l;**

**/\* Generate the string representation, this method produces**

**\* an reversed string. \*/**

**v = (value < 0) ? -value : value;**

**p = s;**

**do {**

**\*p++ = '0'+(v%10);**

**v /= 10;**

**} while(v);**

**if (value < 0) \*p++ = '-';**

**/\* Compute length and add null term. \*/**

**l = p-s;**

**\*p = '\0';**

**/\* Reverse the string. \*/**

**p--;**

**while(s < p) {**

**aux = \*s;**

**\*s = \*p;**

**\*p = aux;**

**s++;**

**p--;**

**}**

**return l;**

**}**

#### sdsull2str

将无符号long long 类型的整数转换成字符串。

**int sdsull2str(char \*s, unsigned long long v) {**

**char \*p, aux;**

**size\_t l;**

**/\* Generate the string representation, this method produces**

**\* an reversed string. \*/**

**p = s;**

**do {**

**\*p++ = '0'+(v%10);**

**v /= 10;**

**} while(v);**

**/\* Compute length and add null term. \*/**

**l = p-s;**

**\*p = '\0';**

**/\* Reverse the string. \*/**

**p--;**

**while(s < p) {**

**aux = \*s;**

**\*s = \*p;**

**\*p = aux;**

**s++;**

**p--;**

**}**

**return l;**

**}**

#### sdsfromlonglong

根据输入的 long long 值 value ，创建一个 SDS。

**sds sdsfromlonglong(long long value) {**

**char buf[SDS\_LLSTR\_SIZE];**

**int len = sdsll2str(buf,value);**

**return sdsnewlen(buf,len);**

**}**

#### sdscatvprintf

打印函数，被 sdscatprintf 所调用。