Redis源码阅读

Server.h 核心数据结构（配置等）

基本数据类型

/\* The actual Redis Object \*/

#define OBJ\_STRING 0

#define OBJ\_LIST 1

#define OBJ\_SET 2

#define OBJ\_ZSET 3

#define OBJ\_HASH 4

在创建字符串的时候会根据实际的大小创建不同类型的字符串

#define SDS\_TYPE\_5 0

#define SDS\_TYPE\_8 1

#define SDS\_TYPE\_16 2

#define SDS\_TYPE\_32 3

#define SDS\_TYPE\_64 4

#define SDS\_TYPE\_MASK 7

#define SDS\_TYPE\_BITS 3

如果是空字符串，实际会使用SDS\_TYPE\_8，因为空字符串创建起来一般是用来添加的

Sds创建起来重视null terminal 的

所以在分配内存的时候多分配的一个用来存放’\0’

sh = s\_malloc(hdrlen+initlen+1);

内存结构如下

|  |  |
| --- | --- |
| Head(len,alloc ,flags) | String(是null terminal的字符串) |

不过SDS\_TYPE\_5的是特殊处理的 flags是用来存类型和长度的，剩下5bit用来存长度

所以最大存储2的5次方长度的字符串

这里所说的长度是不包含null terminal的

/\* Free an sds string. No operation is performed if 's' is NULL. \*/

void sdsfree(sds s) {

if (s == NULL) return;

s\_free((char\*)s-sdsHdrSize(s[-1]));

}

释放sds的时候是将字符串指针回移sds头（具体移动多少根据字符串的类型）

sds使用的头

struct \_\_attribute\_\_ ((\_\_packed\_\_)) sdshdr8 {

uint8\_t len; /\* used \*/

uint8\_t alloc; /\* excluding the header and null terminator \*/

unsigned char flags; /\* 3 lsb of type, 5 unused bits \*/

char buf[];

};

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

size\_t curlen = sdslen(s);

if (len <= curlen) return s;

s = sdsMakeRoomFor(s,len-curlen);

if (s == NULL) return NULL;

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

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

sdssetlen(s, len);

return s;

}

在grow sds的时候，可能会改变sds的类型，如果类型不改变，直接

newsh = s\_realloc(sh, hdrlen+newlen+1);

如果改变类型

/\* Since the header size changes, need to move the string forward,

\* and can't use realloc \*/

newsh = s\_malloc(hdrlen+newlen+1);

if (newsh == NULL) return NULL;

memcpy((char\*)newsh+hdrlen, s, len+1);

s\_free(sh);

s = (char\*)newsh+hdrlen;

s[-1] = type;

sdssetlen(s, len);

sds sdscatvprintf(sds s, const char \*fmt, va\_list ap) {

首先在使用在栈上面分配的1024byte的内存

如果不够用则直接动态分配，每次两倍

并且在内存的buf[buflen-2] = '\0';位置出特殊处理用来判断vsnprintf是否成功

1．OBJ\_STRING sds.h sds.c

typedef char \*sds;

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

/\* Create a new sds string starting from a null terminated C string. \*/

sds sdsnew(const char \*init);

sds sdsempty(void);

sds sdsdup(const sds s);

void sdsfree(sds s);

sds sdsgrowzero(sds s, size\_t len);

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

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

sds sdscatsds(sds s, const sds t);

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

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

sds sdscatvprintf(sds s, const char \*fmt, va\_list ap);

#ifdef \_\_GNUC\_\_

sds sdscatprintf(sds s, const char \*fmt, ...)

\_\_attribute\_\_((format(printf, 2, 3)));

#else

sds sdscatprintf(sds s, const char \*fmt, ...);

#endif

sds sdscatfmt(sds s, char const \*fmt, ...);

sds sdstrim(sds s, const char \*cset);

void sdsrange(sds s, int start, int end);

void sdsupdatelen(sds s);

void sdsclear(sds s);

int sdscmp(const sds s1, const sds s2);

sds \*sdssplitlen(const char \*s, int len, const char \*sep, int seplen, int \*count);

void sdsfreesplitres(sds \*tokens, int count);

void sdstolower(sds s);

void sdstoupper(sds s);

sds sdsfromlonglong(long long value);

sds sdscatrepr(sds s, const char \*p, size\_t len);

sds \*sdssplitargs(const char \*line, int \*argc);

sds sdsmapchars(sds s, const char \*from, const char \*to, size\_t setlen);

sds sdsjoin(char \*\*argv, int argc, char \*sep);

sds sdsjoinsds(sds \*argv, int argc, const char \*sep, size\_t seplen);

/\* Low level functions exposed to the user API \*/

sds sdsMakeRoomFor(sds s, size\_t addlen);

void sdsIncrLen(sds s, int incr);

sds sdsRemoveFreeSpace(sds s);

size\_t sdsAllocSize(sds s);

void \*sdsAllocPtr(sds s);

/\* Export the allocator used by SDS to the program using SDS.

\* Sometimes the program SDS is linked to, may use a different set of

\* allocators, but may want to allocate or free things that SDS will

\* respectively free or allocate. \*/

void \*sds\_malloc(size\_t size);

void \*sds\_realloc(void \*ptr, size\_t size);

void sds\_free(void \*ptr);