Gui内部涉及到大量动态内存分配相关的需要，而常规的内存分配接口并不契合我们的需求，最主要的问题是内外部碎片化，以及内存周期化的管理

**设计参考：**

通常我们将Gui所需分配的元素按类型简单的分为三个类别，并且将对应的内存分配来源管理到一个特定的内存区域，这意味着我们需要涉及到内存管理器以及内存分区，这是可能的三个分区

/\* 内存分配类型:

\* 1.默认类型

\* 特点:碎片化,不稳定生存期,大小不定

\* 建议:使用常规内存分配策略

\* 2.字体类型

\* 特点:碎片化,不稳定生存期,大小稳定

\* 建议:可使用默认类型同一分配器

\* 也可单独开设分配器

\* 3.图形类型

\* 特点:块状化,稳定生存期,大小不限

\* 建议:单独为其开设分配器,唯一化分配

\*/

typedef enum {

scui\_mem\_type\_none = 0,

scui\_mem\_type\_mix,

scui\_mem\_type\_font,

scui\_mem\_type\_graph,

scui\_mem\_type\_num,

} scui\_mem\_type\_t;

最终系统主要使用以下的俩个接口实现内存的分配和释放：

SCUI\_MEM\_ALLOC(type, size)

SCUI\_MEM\_FREE(ptr)

同样的，Gui的内存分配应该具备使用情况的监控以及使用情况的统计，这部分内存通常在各个分区初始化后就分配出来，用于保存内存的分配情况，包括分配地点（哪个文件，哪个行，哪个函数），分配大小，分配类型，分配地址等相关信息，有些地方的内存泄露可能用得着该分析策略，这是可能的一个实现方式

void scui\_mem\_record\_analysis(bool force);

void scui\_mem\_record\_statistic(bool force);

**内存分区参考**：

#if 0

#elif SCUI\_MEM\_SIZE\_TYPE == 0 // 4M PSRAM (空闲1M)

#define SCUI\_MEM\_TYPE\_SIZE\_MIX (1024 \* 128)

#define SCUI\_MEM\_TYPE\_SIZE\_FONT (1024 \* 128)

#define SCUI\_MEM\_TYPE\_SIZE\_GRAPH (1024 \* 1024 \* 3 - 1024 \* (128 + 128))

#elif SCUI\_MEM\_SIZE\_TYPE == 1 // 8M PSRAM (空闲1M)

#define SCUI\_MEM\_TYPE\_SIZE\_MIX (1024 \* 256)

#define SCUI\_MEM\_TYPE\_SIZE\_FONT (1024 \* 512)

#define SCUI\_MEM\_TYPE\_SIZE\_GRAPH (1024 \* 1024 \* 7 - 1024 \* (256 + 512))

#else

#error "unknown type"

#endif

**可供参考的内存管理器**:

双端内存分配策略：

分配规则：内存从头部和尾部向中心生长，用于将热数据和冷数据做隔离，在混合内存分配策略中，对一个类型的分配目标，根据使用场景进行进一步的分开

使用参考：图形相关的资源可能发挥意想不到的效果，将双帧缓冲在开始时分配，沉积在头部，画布接续在之后分配，而图片相关的内存则可以逆向分配，这样子就可以降低碎片的风险，尽可能保证在大内存申请时的成功率

一级内存分配策略：参考移植源码zephyr -> heap

这可以作为通用选择，如果不知道如何选取内存管理器，可以无脑用这个

二级内存分配策略：参考移植源码lvgl -> tlsf

不太建议使用，该分配器主要原因remalloc被大面积使用时考虑，gui应该尽可能减少对此类接口的使用，尽可能采用静态生长策略