一.在Lua5.1.4版本中TString的实现如下:

```
typedef union TString {
  L_Umaxalign dummy;  /* ensures maximum alignment for strings */
  struct {
    CommonHeader;
    lu_byte reserved;
    unsigned int hash;
    size_t len;
  } tsv;
} TString;
```

- dummy : 用于对齐
- CommonHeader : gc对象的标准头
- reserved : 是否是Lua虚拟机的保留字段(关键字),如果为1,则不会被gc回收
- hash : 字符串的散列值
- len : 字符串长度

在Lua5.3.4版本中TString的实现如下:

```
typedef struct TString {
  CommonHeader;
  lu_byte extra;  /* reserved words for short strings; "has hash" for longs */
  lu_byte shrlen;  /* length for short strings */
  unsigned int hash;
  union {
    size_t lnglen;  /* length for long strings */
    struct TString *hnext;  /* linked list for hash table */
  } u;
} TString;
```

- CommonHeader
- extra: luaX_init中使用到这个, 用来表示是关键字类别.
- shrlen: 短字符串的长度
- hash: hash值
- u.lnglen: 长字符串的长度
- u.hnext: 如果出现hash碰撞, 则将碰撞内容链接到链表内

二.Lua5.1.4的实现中无论对于长字符串还是短字符串都是使用的stringtable来存入散列桶

```
typedef struct global_State {
  stringtable strt;  /* hash table for strings */
  ...
}
```

Lua5.3.4的实现中对于长字符串是单独存储,并使用strcache来缓存,对于短字符串还是像原来一样存储在stringtable中.

```
//lstring.c
#define LUAI_MAXSHORTLEN   40

TString* luaS_newlstr (lua_State *L, const char *str, size_t l) {
  if (l <= LUAI_MAXSHORTLEN)  /* short string? */
    return internshrstr(L, str, l);
  else {
    TString *ts;
    if (l >= (MAX_SIZE - sizeof(TString))/sizeof(char))
      luaM_toobig(L);
    ts = luaS_createlngstrobj(L, l);
    memcpy(getstr(ts), str, l * sizeof(char));
    return ts;
  }
}

typedef struct global_State {
  ...
  stringtable strt;  /* hash table for strings */
  ...
  TString *strcache[STRCACHE_N][STRCACHE_M];  /* cache for strings in API */
} global_State;
```

三.Lua5.1.4的实现中字符串的对比是依靠对比gc指针是否相同来判断的

```
#define gcvalue(o) check_exp(iscollectable(o), (o)->value.gc)
//lvm.c
int luaV_equalval (lua_State *L, const TValue *t1, const TValue *t2) {
  const TValue *tm;
  lua_assert(ttype(t1) == ttype(t2));
  switch (ttype(t1)) {
    case LUA_TNIL: return 1;
    case LUA_TNUMBER: return luai_numeq(nvalue(t1), nvalue(t2));
    case LUA_TBOOLEAN: return bvalue(t1) == bvalue(t2);  /* true must be 1 !! */
    case LUA_TLIGHTUSERDATA: return pvalue(t1) == pvalue(t2);
    case LUA_TUSERDATA: {
      if (uvalue(t1) == uvalue(t2)) return 1;
      tm = get_compTM(L, uvalue(t1)->metatable, uvalue(t2)->metatable,
                      TM_EQ);
      break;  /* will try TM */
    }
    case LUA_TTABLE: {
      if (hvalue(t1) == hvalue(t2)) return 1;
      tm = get_compTM(L, hvalue(t1)->metatable, hvalue(t2)->metatable, TM_EQ);
      break;  /* will try TM */
    }
    default: return gcvalue(t1) == gcvalue(t2);  /* here */
  }
  if (tm == NULL) return 0;  /* no TM? */
  callTMres(L, L->top, tm, t1, t2);  /* call TM */
  return !l_isfalse(L->top);
}
```

Lua5.3.4的实现中字符串的对比分长字符串和短字符串,
对于短字符串依旧和原来的逻辑类似 :

```
#define tsvalue(o) check_exp(ttisstring(o), gco2ts(val_(o).gc))
#define eqshrstr(a,b) check_exp((a)->tt == LUA_TSHRSTR, (a) == (b))
//lvm.c
switch (ttype(t1)) {
    ...
    case LUA_TSHRSTR: return eqshrstr(tsvalue(t1), tsvalue(t2));
    ...
}
```

对于长字符串来说,因为长字符串可能不止一个副本,需要使用memcmp来比较:

```
//lstring.h
int luaS_eqlngstr (TString *a, TString *b) {
  size_t len = a->u.lnglen;
  lua_assert(a->tt == LUA_TLNGSTR && b->tt == LUA_TLNGSTR);
  return (a == b) ||  /* same instance or... */
    ((len == b->u.lnglen) &&  /* equal length and ... */
     (memcmp(getstr(a), getstr(b), len) == 0));  /* equal contents */
}
```