zebra app中 log相关设置，不同的应用中

struct zlog

ident ： log来源，为程序名，传入openlog
protocol： Zlog协议，比如RIPNG_LOG等等
flags：表明log输出到哪儿，初始设置为ZLOG_NOLOG
fp：文件指针
filename：文件名指针
syslog：
stat：
connected：
maskpri：as per syslog setlogmask
priority：as per syslog priority
facility： as per syslog facility需要传递给openlog的，使用值为LOG_DAEMON
record_priority：初始设置值为0

类型

<<枚举： start = 0>>
zlog_proto_t

ZLOG_NONE
ZLOG_DEFAULT
ZLOG_ZEBRA
ZLOG_RIP
ZLOG_BGP
ZLOG_OSPF
ZLOG_RIPNG
ZLOG_OSPF6
ZLOG_MASC

类型

<<常量>>
flags 实际上使用define

#define ZLOG_NOLOG 0x00
#define ZLOG_FILE 0x01
#define ZLOG_SYSLOG 0x02
#define ZLOG_STDOUT 0x04
#define ZLOG_STDERR 0x08

<<define>>
预编译宏

HAVE_IPV6 控制IPv6 【enable】
HAVE_IRDP 【disable】
HAVE_SNMP [0]
HAVE_SIN_LEN
HAVE_RUSAGE [1]
HAVE_SUN_LEN
HAVE_SOCKADDR_DL
HAVE_TCP_ZEBRA Use TCP for zebra communication
HAVE_NETLINK [1]

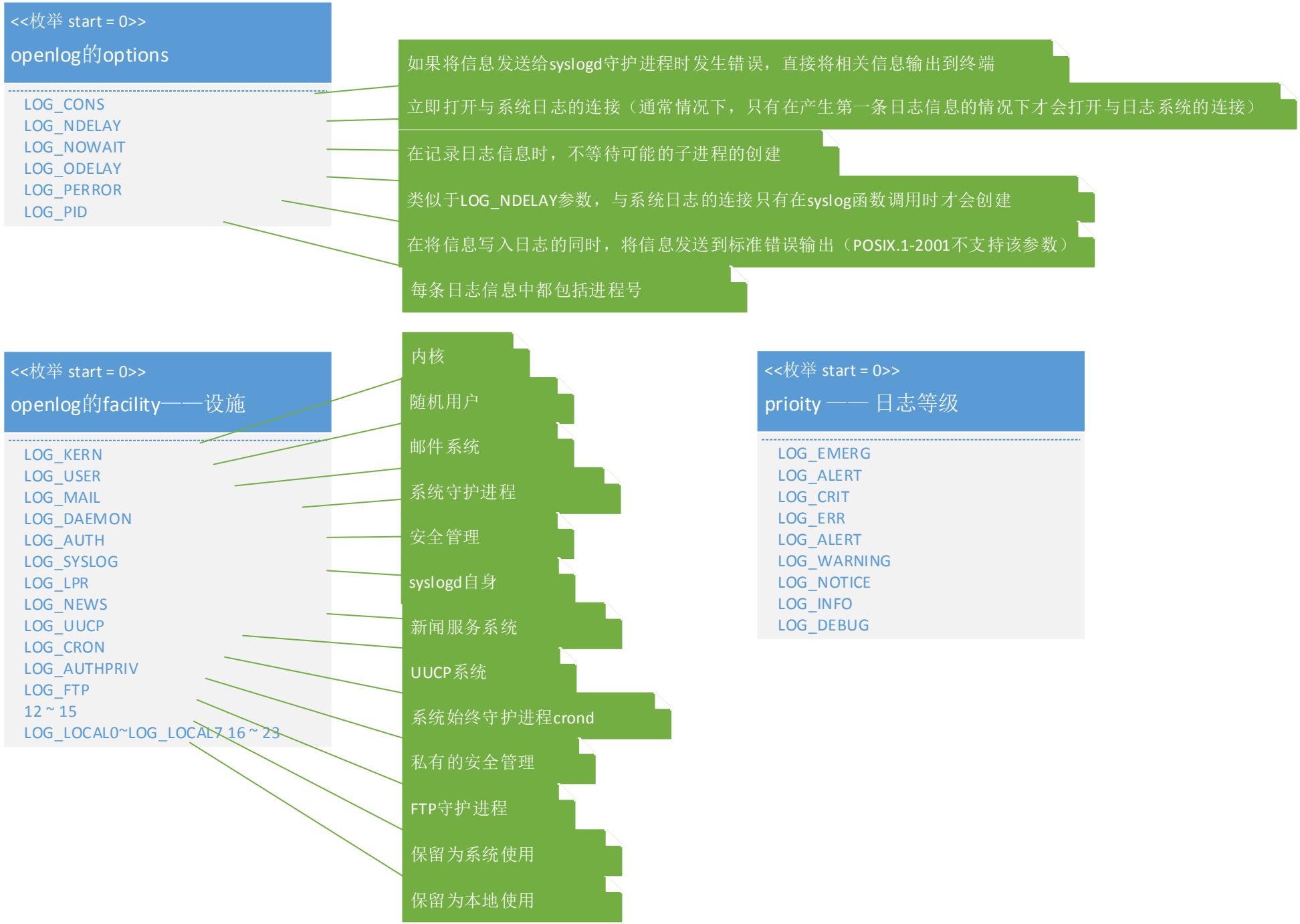
AGGREGATE_NEXTHOP_CHECK
IOV_MAX
SA_RESTART
BUILD_STATIC
RTADV
NBMA_ENABLE
RIP_RECVMSG
TIMER_NO_SORT
THREAD_CONSUMED_TIME_CHECK
VTYSH [0]
SUNOS_5 [0]
BSDI_NRL
SIN6_LEN

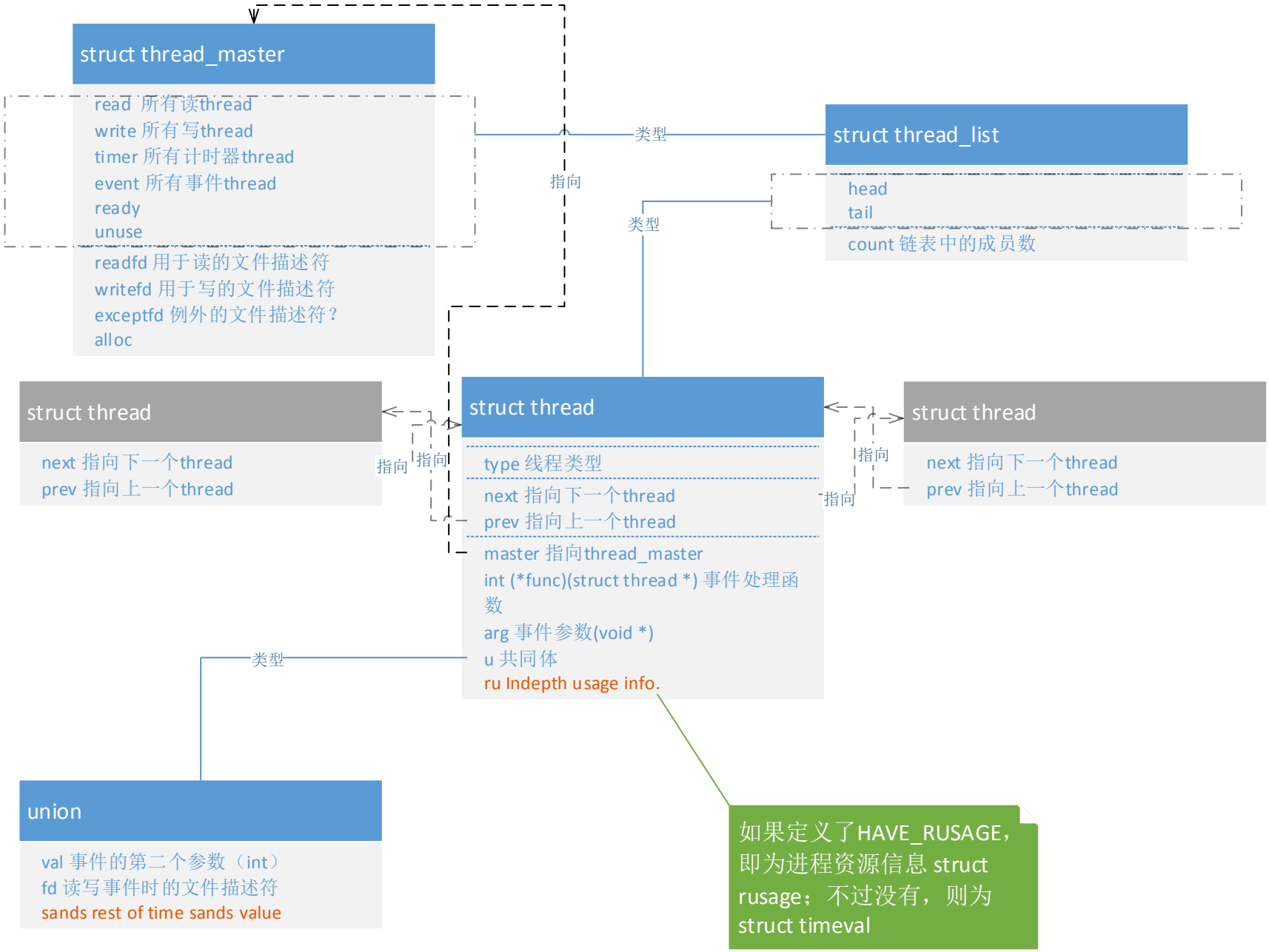
TP_CMD_SUPPORT
TP_RIP_DEBUG
TP_LIST_SUPPORT
TP_AUTH_SUPPORT
TP_SUPPORT

MEMORY_LOG
TELNET_OPTION_DEBUG
DEBUG
VTYSH_DEBUG

除TP_开头的宏，在config.h.in中均能找到功能解释

日志输出形式：*<facility<<3|priority>ident:formatted string*




struct thread_master
read 所有读thread
write 所有写thread
timer 所有计时器thread
event 所有事件thread
ready
unuse
readfd 用于读的文件描述符
writefd 用于写的文件描述符
exceptfd 例外的文件描述符？
alloc
类型
struct thread_list
head
tail
count 链表中的成员数
指向
类型
struct thread
next 指向下一个thread
prev 指向上一个thread
指向
指向
struct thread
type 线程类型
next 指向下一个thread
prev 指向上一个thread
master 指向thread_master
int (*func)(struct thread *) 事件处理函数
arg 事件参数(void *)
u 共同体
ru Indepth usage info.
指向
指向
struct thread
next 指向下一个thread
prev 指向上一个thread
类型
union
val 事件的第二个参数（int）
fd 读写事件时的文件描述符
sands rest of time sands value
如果定义了HAVE_RUSAGE，即为进程资源信息 struct rusage；不过没有，则为 struct timeval

struct host

name 路由器host name
password 虚拟终端vty的密码
password_encrypt
enable 开关认证，需要使用密码
enable_encrypt 开关密码加密
lines 系统终端行宽？

logfile 日志文件
log_stdout 日志标准输出(u_char)
log_syslog 日志syslog(u_char)

config 路由器host配置文件
advanced 服务flags(int)
encrypt 服务flags(int)
motd Banner Configuration(char *)

初始化：
lines = -1;
motd = 一段描述信息;
其余为NULL/0

struct cmd_node

node node类型，也是在cmdvec中的index
vtysh Is this node's configuration goes to vtysh ? (int)
prompt 虚拟终端vty提示字符
func 该node操作配置文件写函数int (*)(struct vty *)
cmd_vector 该节点的command数组

类型

<<枚举 start = 0>>
node_type

AUTH_NODE
VIEW_NODE
…

vector(struct _vector)

max 已经使用slot的数目
alloced 申请到的slot数目
index 数据索引(void **)

指向

数据指针数组，成员为void *

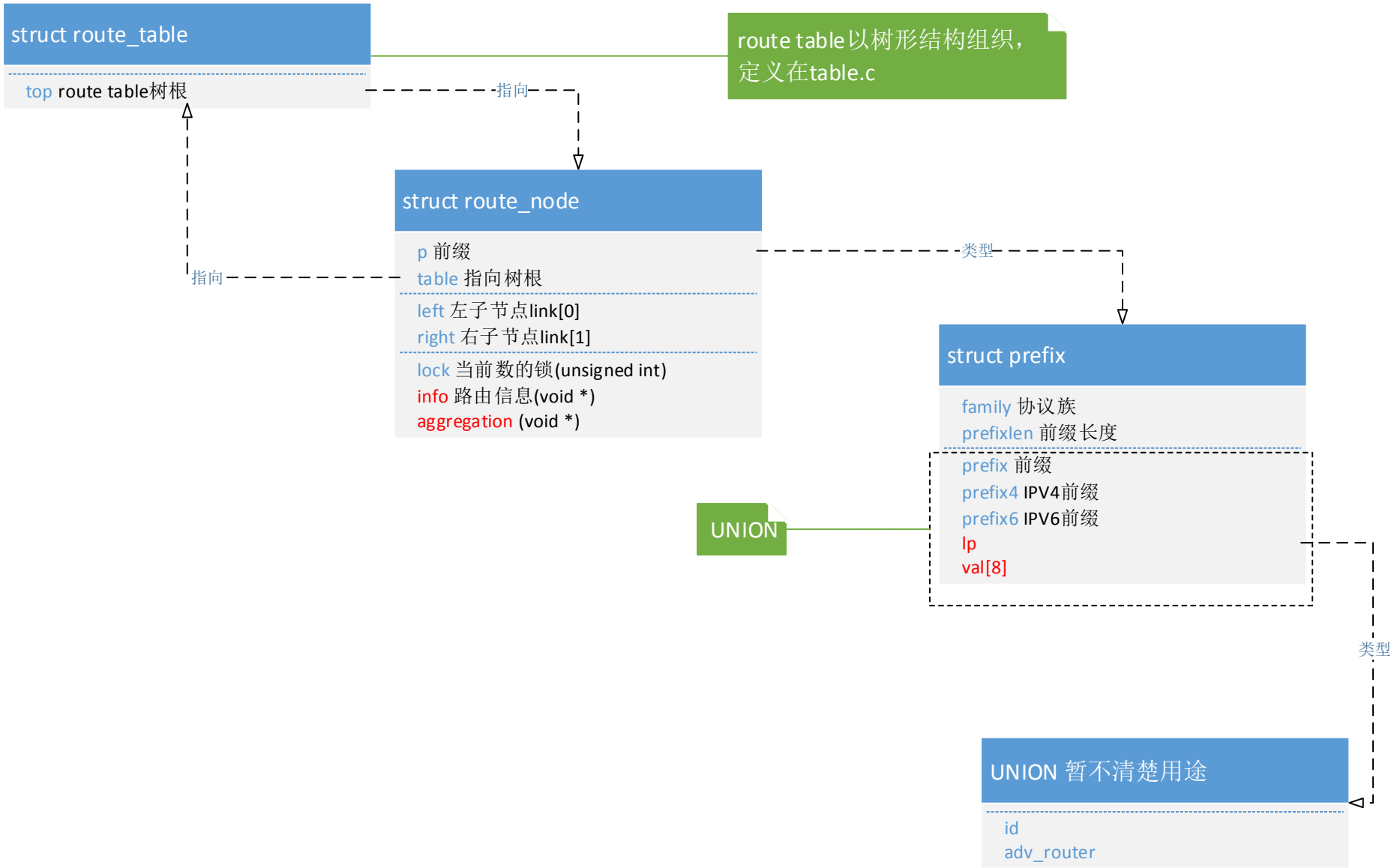
[0]
[1]
[2]
…
[alloced - 1]

vector的实现很有意思：
（1）初始化时，alloced = 1，max = 0；
（2）使用时如果不够，则倍增alloced *= 2，直到比需要的多为止；
（3）max用来表示当前使用的slot数量；
（4）因为数据为指针void *，也是说每个成员的数据大小是可变的；
具体实现在vector.c中，完全可以拿来使用。

struct cmd_element

string 命令command标识符(char *)
int (*func)(cmd_element*, vty*, int, char**)
doc 命令说明documentation(char *)
daemon 命令属于哪个模块(int)
strvec 描述数组(vector
cmdsize Command index count
config 配置信息(char *)
subconfig 子配置信息数组(vector)

类型


struct route_table
top route table树根
指向
route table以树形结构组织，
定义在table.c
struct route_node
p 前缀
table 指向树根
left 左子节点link[0]
right 右子节点link[1]
lock 当前数的锁(unsigned int)
info 路由信息(void *)
aggregation (void *)
指向
类型
struct prefix
family 协议族
prefixlen 前缀长度
prefix 前缀
prefix4 IPV4前缀
prefix6 IPV6前缀
lp
val[8]
UNION
类型
UNION 暂不清楚用途
id
adv_router

struct ripng

- sock RIPNG描述符
- command RIPNG command用于确定报文的command字段
- version RIGNG version，同报文
- update_time 更新时间
- timeout_time 老化时间
- garbage_time 垃圾回收时间
- max_mtu 最大mtu
- default_metric 默认metric值(int)
- default_information
- ibuf 输入buffer
- obuf 输出buffer
- table RIPNG路由信息树(route_table)
- route RIPNG静态路由信息树
- aggregation RIPNG 聚合路由信息树
- t_read 读线程(struct thread *)
- t_write 写线程
- t_update 更新线程
- t_garbege 垃圾回收线程
- t_zebra 服务zebra线程
- trigger 是否需要级联更新(int)
- t_triggered_update 级联更新线程
- t_triggered_interval

类型

struct steam_fifo

- count 队列大小
- head 队列头
- tail 队列尾

类型

struct stream

- next 下一个stream指针
- data 数据指针
- putp 写位置（unsigned long）
- getp 读位置
- endp 结束位置
- size 数据大小

struct ripng_packet

- command 区分request/response
- version RIPNG版本
- zero 必须为0的字段
- rte[1] route信息entry

元素类型

定义结构体时，声明数组长度为1，实际将后面所有数据均为数组成员

struct rte

- addr 目的IPV6地址
- tag tag字段(u_short)
- prefixLen 前缀长度
- metric 距离

struct distribute

- ifname 属于哪一个interface
- list[2] access_list的name
- prefix[2] prefix_list的name

所有对象均保存在disthash中，类似于ifrmaphash与ifr_map的关系

类型

struct access_list

- name
- remark
- master
- type
- next
- priv
- head
- tail

struct access_list

- ...
- next
- priv
- ...

struct access_list

- ...
- next
- priv
- ...

类型

struct access_master

- num
- str
- void (*add_hook)()
- void (*delete_hook)()

类型

<<枚举 start = 0>>
access_type

- ACCESS_TYPE_STRING
- ACCESS_TYPE_NUMBER

struct ripng_interface

- enable_network 是否启用RIPng
- enable_interface 打开关闭此interface
- running 是否正在运行RIPng
- list[RIPNG_FILTER_MAX = 2] access-list
- prefix[RIPNG_FILTER_MAX] prefix_list , 应该是和access_list一一对应的
- routemap[RIPNG_FILTER_MAX] routemap，应该也是和前两者一一对应
- rtag RIPng tag配置
- default_originate 是否产生默认路由信息
- default_only 只包括默认路由信息
- t_wakeup 唤醒线程
- passive passive接口

struct access_list_list

- head
- tail

struct filter

- next
- prev
- ...

struct filter

- next
- prev
- type
- cisco
- cfilter
- zfilter

struct filter

- next
- prev
- ...

<<枚举 start = 0>>
filter_type

- FILTER_DENY
- FILTER_PERMIT
- FILTER_DYNAMIC

类型

struct prefix_list

- name 名字
- desc 描述
- master
- type
- count
- rangecount
- head
- tail
- next
- prev

<<枚举 start = 0>>
prefix_name_type

- PREFIX_TYPE_STRING
- PREFIX_TYPE_NUMBER

struct ripng_tag

- tag
- port
- maddr
- table
- distance
- split_horizon
- poison_reserve

struct prefix_master

- num
- str
- seqnum
- recent
- void (*add_hook)()
- void (*delete_hook)()

struct prefix_list_entry

- seq
- le
- ge
- type
- any
- prefix
- refcnt
- hitcnt
- next
- prev

<<枚举 start = 0>>
prefix_list_type

- PREFIX_DENY
- PREFIX_PERMIT

struct prefix_list_list

- head
- tail

struct route_map

- name
- head
- tail
- next
- prev

struct route_map_index

- map
- pref
- type
- exitpolicy
- nextpref
- match_list 匹配列表，类似于iptables match
- set_list 如果匹配了，这是动作list
- next
- prev

<<枚举 start = 0>>
route_map_type

- RMAP_PERMIT
- RMAP_DENY
- RMAP_ANY

<<枚举 start = 0>>
route_map_end_t

- RMAP_EXIT 匹配后直接退出
- RMAP_GOTO 继续匹配、直到pref的值大于当前的nextpref值
- RMAP_NEXT 继续下一个

route_map_master

struct route_map_list

- head
- tail
- add_hook
- delete_hook
- event_hook

struct route_map_rule_list

- head
- tail

struct route_map_rule_cmd

- str
- func_apply 匹配、设置函数
- func_compile
- func_free

该函数原型，是一种典型的接口编程：

```
route_map_result_t (*func_apply)(void *, struct prefix *,
                                 route_map_object_t, void *);
```

其中，输出route_map_result_t：

```
{
 RMAP_MATCH,
 RMAP_DENYMATCH,
 RMAP_NOMATCH,
 RMAP_ERROR,
 RMAP_OKAY
}
```

而输入参数route_map_object_t

```
{
 RMAP_RIP,
 RMAP_RIPNG,
 RMAP_OSPF,
 RMAP_OSPF6,
 RMAP_BGP
}
```

该函数位于match_list，则用于匹配；
该函数位于set_list，则用于匹配后的操作

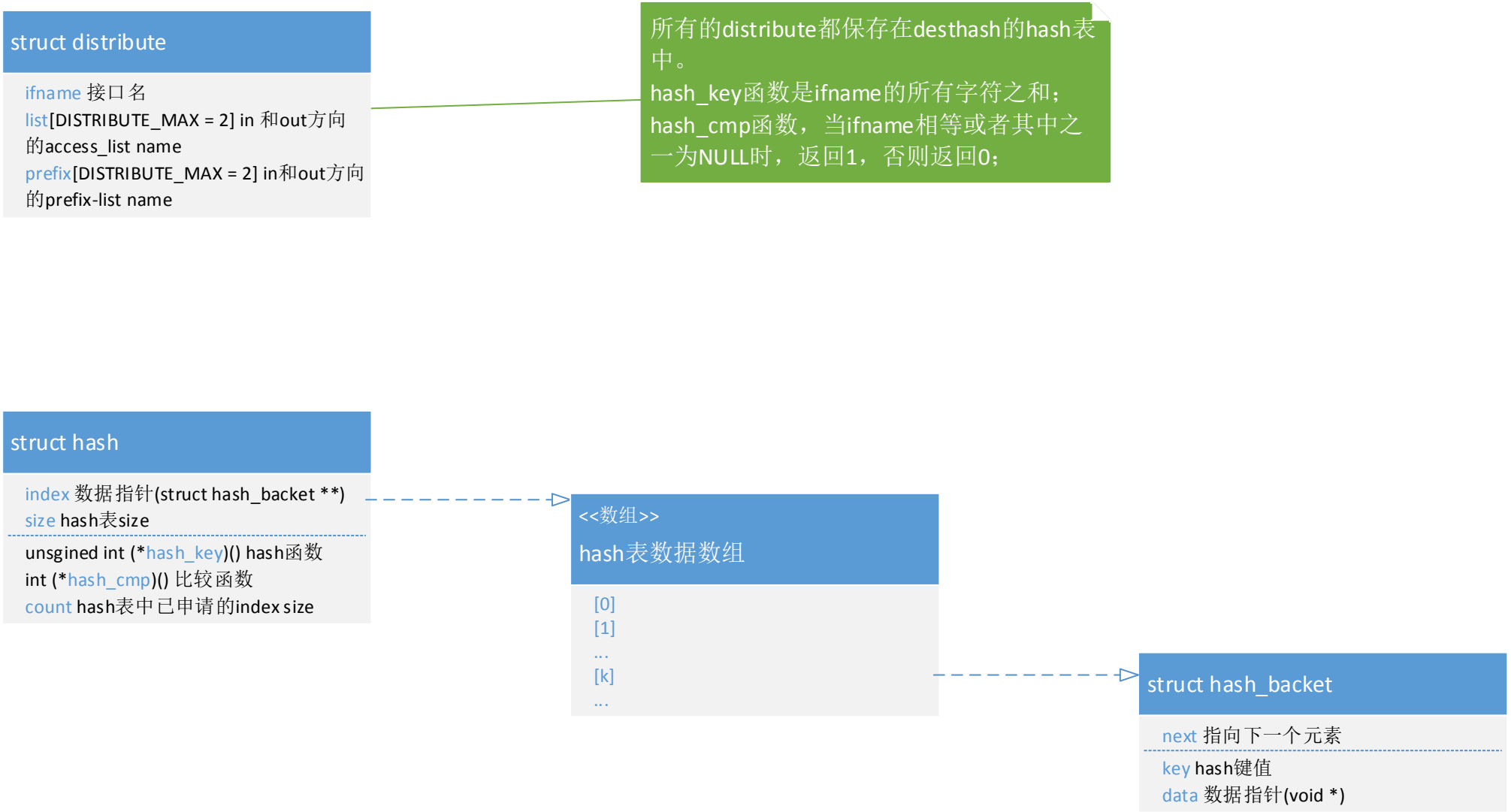
struct route_map_rule

- cmd
- rule_str
- value
- next
- prev

struct if_rmap

- ifname
- routemap[2] 对应route_map的名字

该结构是route_map的封装结构，所有（不论是否正在使用）的route_map对应的if_rmap结构保存在ifrmaphash中


struct distribute
ifname 接口名
list[DISTRIBUTE_MAX = 2] in 和out方向的access_list name
prefix[DISTRIBUTE_MAX = 2] in和out方向的prefix-list name
所有的distribute都保存在desthash的hash表中。
hash_key函数是ifname的所有字符之和；
hash_cmp函数，当ifname相等或者其中之一为NULL时，返回1，否则返回0；
struct hash
index 数据指针(struct hash_backet **)
size hash表size
unsgined int (*hash_key)() hash函数
int (*hash_cmp)() 比较函数
count hash表中已申请的index size
<<数组>>
hash表数据数组
[0]
[1]
...
[k]
...
struct hash_backet
next 指向下一个元素
key hash键值
data 数据指针(void *)