技 术 文 件

技术文件名称：NGN PPLAT SIP V1.01.10_详细设计说明

技术文件编号：

版　　本：V1.01.10

共 57 页

(包括封面)

拟　制	SIP 平台项目组
审　核	苏登军
会　签	张永智
	张政山
	任鹏
	管建池
标准化	黄纯娟
批　准	丁学新

中兴通讯股份有限公司

修改记录

说明：

发起的项目包括：CSCF，MGCF，PSS，PPLAT；

影响的项目选择包括：CSCF，MGCF，PSS；

修订编号	文件编号	版本号	拟制/修改人	拟制/修改日期	更改理由	主要更改内容（写要点即可）	发起的项目	影响的项目	合入版本	合入时间
		V1.01.10	管建池	2008/07/22	无	无	PSS			
0001		V1.01.10	张志海	2009/01/05	增加	6.1.1 添加与修改 record0route 描述	CSCF	PSS		
0002		V1.01.10	张志海	2009-01-05	增加	增加 11 章兼容性说明	PPLAT			
0003		V1.01.10	曾丽君	2009-05-22	增加	6.1.2 record route 优化	PSS			

目 录

1 编写目的

本文描述对象为 NGN PPLAT SIP 协议平台 V1.01.10 版本详细设计，通过对设计的思路，函数，数据的详细描述。以满足上游文档《NGN PPLAT V1.01.10_系统方案》中对的要求。同时为本模块的编码，单元测试和集成测试等工作提供依据。以利于程序员编制程序。

2 术语、定义和缩略语

2.1 术语、定义

文使用的专用术语、通用术语、定义见《NGN PPLAT V1.00.10_SIP 协议 术语、定义和缩略语》。

2.2 缩略语

本文使用的通用缩略语见《NGN PPLAT V1.00.10_SIP 协议术语、定义和缩略语》。

3 设计依据

本文的设计依据见表 3.1。

表 3.1

文件编号	文件名称	版本号	说明
	NGN PPLAT V1.01.10_系统方案	V1.0	上游文档
	NGN PPLAT V1.00.10_SIP 协议术语、定义和缩略语	V1.0	引用文档

4 概述

在 V1.01 版本中，主要做了两件事情一个是对 ATCA 的支持，一个是 UA 模式下对于 1xx 和 2xx 重传与吸收处理。

其中为了支持 ATCA，V1.01 版本采用了进程多实例方案。为此引入了两个新的进程：TU 管理进程和 TP 管理进程。其中 TU 管理进程主要负责在收到上下行消息时，通过策略选择处理进程后进行分发，从而实现不同处理进程间的负荷均衡。

TP 管理层的功能也主要是在收到上下行消息时，通过策略选择处理进程后进行分发，从而实现不同处理进程间的符合均衡。特别的是，作为其均衡控制角色的一部分，有些时候 TP 管理进程需要从承载接收消息或者向承载发送消息，这就使得 TP 管理进程需要有创建和维护链路、SOCKET 的功能，同时由于 IPSec 和 Socket 的紧密绑定关系，因此和 TP 管理进程创建的 SOCKET 相关的 IPSec 流程也需要在 TP 管理进程处理。TP 管理进程和 TP 处理进程的分界是 TP 管理进程只负责消息的接收和转发，它从 SOCKET 里读取报文，以文本的形式转发给 TP 处理进程处理，并不负责将文本转换成 TLV 格式，它从 TP 管理进程接收到的需要发送的报文也是文本格式的，并不负责将 TLV 格式报文转换成文本格式。

另外为了处理来自 TP 管理进程的上行消息，TP 处理进程需要增加对来自 TP 管理进程上行消息的处理。同时在发送消息时，如果发送消息时使用的 SOCKET 不在本板创建则转发至 TP 管理进程。

同时在 ATCA 中对于进程的标识已经发生了变化，对于这一变化的处理也是本文关注的内容。

为了支持对 1xx 和 2xx 的重传和吸收，需要修改 TU 处理进程对于在 UA 模式下对于上

下行 1xx 和 2xx 响应的处理。

在 V1.01 还提供了一个相对独立的功能---内存分配器，该功能主要用在编解码上用以协助编解码创建多实例。

另外在 V1.01 增加了对于处理网管发送的跟踪类任务的设置与取消功能。

5 标准模块

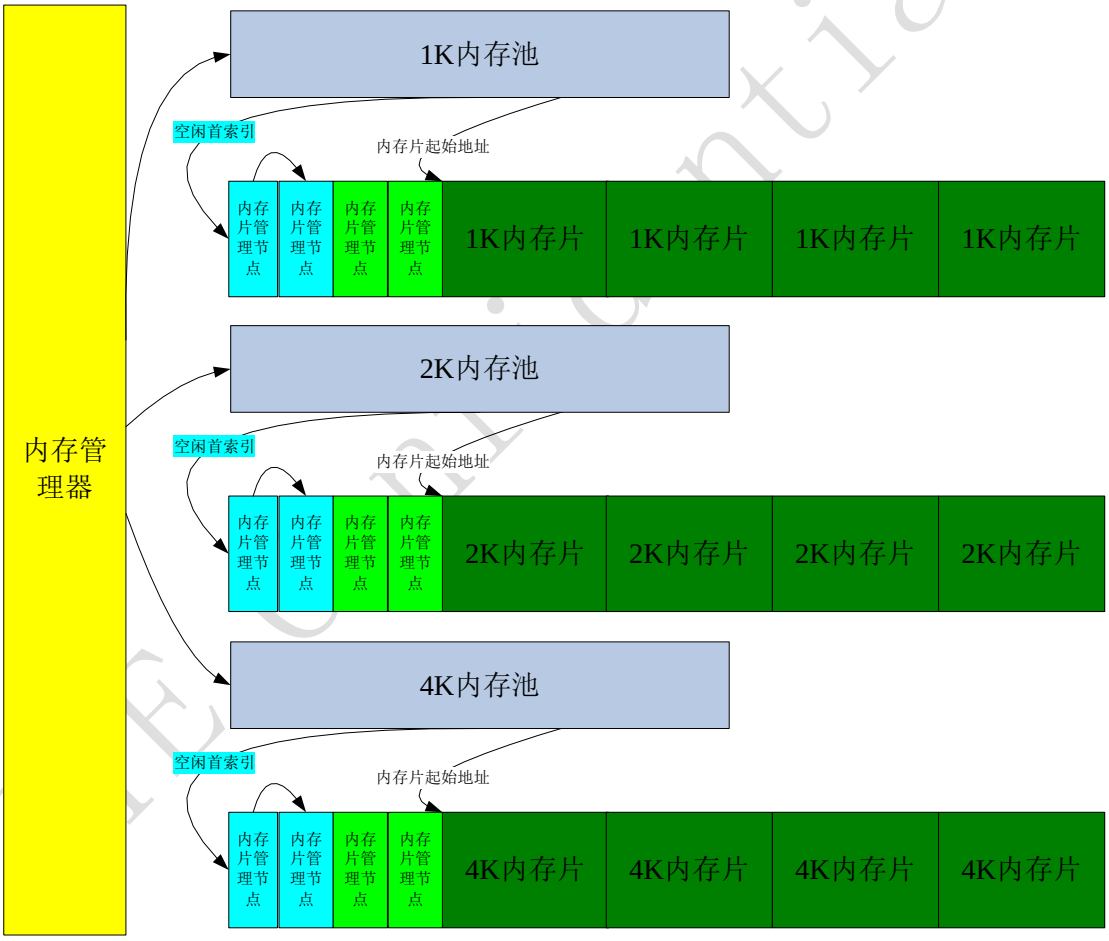
未引用任何已有的标准模块，也未打算提供新的标准模块。

6 模块设计

6.1 实现原理

6.1.1 内存分配器

内存分配器的实现原理如下图：



内存分配器管理的内存资源被分为两部分，一部分用于内存篇管理节点，一部分用于内存片，内存片管理节点和内存片一一对应，内存管理节点内维护着对应内存片的占用状态。空闲的内存片管理节点串连在一起形成一个单向链表。

内存分配器采用分级管理，等级初步分为 1K，2K，4K，8K，16K，32K 等六个内存池，各等级内存池内内存片的个数由调用者在创建内存分配器时指定。

内存分配器管理的内存资源可以由外部传入也可以由内存分配器自行申请，这由调用这

在创建内存分配器时指定。在创建内存分配器时，如果指示内存由外部传入，而传入的内存又不足够用于内存分配器创建，则内存分配器创建失败；如果指示内存由内存分配器自行申请，而内存分配器在向系统申请内存的时候失败，则内存分配器创建失败。

内存分配器在分配内存时，根据需要分配内存的大小选择合适的内存池，将内存池内空闲内存片管理节点链表的首节点对应的内存片分配给调用者使用，将对应的管理节点占用状态置为已占用，同时将链表首节点从链表中移除，更新内存分配器内保存的链表首节点。如果小的内存池已经没有空闲的内存片，则选择更大级别的内存池，如果都没有空闲，则分配失败。

内存分配器在释放内存时，跟据传入的内存片指针判断属于那个级别的内存池。然后根据内存指针计算出对应的内存片管理节点，将对应节点的占用状态置为空闲，并将该节点加入到空闲链表首，同时更新内存分配器内保存的空闲链表首索引；如果内存指针不在内存片的边界点上，则为非法指针，内存释放失败。

在销毁内存分配器时，如果内存分配器管理的内存由内存分配器自行申请，则销毁时需要将申请的内存释放；否则话，销毁时不进行释放，由外部负责管理内存的释放

6.1.2 1xx 重传

在 UA 模式下，TU 收到业务传下来的 1xx 响应。判断接口消息中的重传标志。如果不需要重传，处理方式和 V1.0 版本一致。如果需要重传，处理如下

1. TU 数据区初始化调用 SIP_TU_CacheList_Init 初始化缓存队列列表
2. UA 方式下收到业务的 INVITE 1xx 消息，调用 SIP_TU_CacheList_AllocCache 申请缓存，并保存 TLV 消息到缓存中去，设置重发定时器，将 bListIdx 存入到 Leg 数据区 b2xxMsgBufIdx
3. 定时器超时，调用 SIP_TU_CacheList_ge tData 获取缓存使用
4. 64*T1 超时或收到 PRACK，调用 SIP_TU_CacheList_FreeCache 释放缓存，置 b2xxMsgBufIdx 为 TU_BYTE_INVALID
5. TU 数据区释放调用 SIP_TU_CacheList_FreeAll

6.1.3 2xx 重传

在 UA 模式下，TU 收到业务传下来的 2xx 响应。判断接口消息中的重传标志。如果不需要重传，处理方式和 V1.0 版本一致。如果需要重传，处理如下

1. TU 数据区初始化调用 SIP_TU_CacheList_Init 初始化缓存队列列表
2. UA 方式下收到业务的 INVITE 2xx 消息，调用 SIP_TU_CacheList_AllocCache 申请缓存，并保存 TLV 消息到缓存中去，设置重发定时器，将 bListIdx 存入到 Leg 数据区 b2xxMsgBufIdx
3. 定时器超时，调用 SIP_TU_CacheList_ge tData 获取缓存使用
4. 64*T1 超时或收到 PRACK，调用 SIP_TU_CacheList_FreeCache 释放缓存，置 b2xxMsgBufIdx 为 TU_BYTE_INVALID
5. TU 数据区释放调用 SIP_TU_CacheList_FreeAll

6.1.4 1xx 和 2xx 重传冲突处理

在 UA 模式下，如果 TU 在收到需要重传的 1xx 响应后且还没有收到对应的 PRACK，又收到新的需要重传的 1xx 响应，则将该响应丢弃。同时向上层业务报错。

在 UA 模式下，如果 TU 在收到需要重传的 1xx 响应后且还没有收到对应的 PRACK，

又收到了初始请求的 2xx 响应。则杀掉 1xx 响应的重传定时器，释放缓存的 1xx 响应。同时负责 2xx 的重传。

6.1.5 重传 1xx 的吸收

在 UA 模式下，TU 层在收到事务层传递上来的 1xx 响应时，如果通过 Rseq 和 CSeq 进行匹配确定是重传的 1xx 响应，调用 DB 接口读取配置判断是否需要吸收重传的 1xx 响应。如果是，则直接丢弃 1xx 响应(PRACK 的重传有 PRACK 事务负责)；如果不需要吸收，则上报至上层应用。

6.1.6 重传 2xx 的吸收

在 UA 模式下，TU 收到业务传下来的 ACK 消息。读取 DB 接口判断是否需要吸收重传的 2xx 响应，如果不需要，则 TU 不负责吸收重传的 2xx 响应，也不负责 ACK 的发送。如果需要，其处理如下：

1. TU 数据区初始化调用 SIP_TU_CacheList_Init 初始化缓存队列列表
2. 上层应用发送 ACK 消息到 TU，TU 在收到 ACK 消息后调用 SIP_TU_CacheList_AllocCache 申请缓存，并保存 TLV 到该缓存，并保存缓存消息的 ID 到 TU 数据区 bAckMsgBufIdx
3. 收到对端重发的 Invite 2xx 消息则通过调用 SIP_TU_CacheList_getData 获取缓存，将缓冲区中的 Ack 消息重发到 TP 层
4. 64T1 超时或收到对端其他请求，调用 SIP_TU_CacheList_FreeCache 释放缓存，置 bAckMsgBufIdx 为 TU_BYTE_INVALID
5. TU 数据区释放调用 SIP_TU_CacheList_FreeAll

6.1.7 对 V1.0 中 PID 结构直接使用的隔离

在 SIPV1.0 中对于 PID 结构的使用体现在如下几个方面

（1） 在 IPSec Handle、CID Handle、DIDhandle 中定义了 Module 号，并且使用这些 module 来拼装 PID，上述结构对上层应用直接暴露，在 ATCA 的实现中 Handle 内的 Module 号被替换为进程标识。由于上层应用对于上述 Handle 的使用是透明的，因此该修改并不影响兼容性。

（2） 在 SIP 平台封装的 DB 接口中定义了 Module 号并且使用这些 module 来拼装 PID，在 ATCA 的实现中 Module 号被替换为进程标识。出于兼容性考虑，SIP 平台通过编译宏来隔离新旧结构的差异。

（3） 在 SIP 平台内部的对 PID 的拼装，由于 ATCA 中与 PID 对应的 JID 结构与 PID 不同，出于兼容性考虑。在 SIP 平台内部拼装 PID 的地方采用编译宏来隔绝与不同平台对应的实现。

（4） 在 SIP 与业务接口中以 PPLAT_SAP_ID_T 方式中的进程号和模块号方式暴露的，上层应用使用这些来拼装 SIP 的 PID。在 ATCA 平台中上述进程号和模块号被替换为 PPLAT_SIP_PID_T，具体接口修改可参见接口说明书。出于兼容性考虑，SIP 平台通过编译宏来隔离新旧结构的差异。如果上层应用与 V1.01 版本运行的在 ATCA 平台上需要对此进行同步修改。

6.1.8 SIP TP 管理进程实现原理

SIP TP 管理进程主要负责上下行消息的分发，其分发的策略由具体项目决定。当在某些配置情况下，SIP TP 管理进程需要从承载收包或者向承载发包，这种情况下，SIP TP 管理进程需要负责链路和 SOCKET 的维护，这部分功能和 SIP TP 对应的功能类似。另外如果此时还要求支持 IPSec 功能，那么和对应 SOCKET 相关的 IPSec 功能也要求 SIP TP 管理进程实现。

SIP TP 管理进程的基本分发场景如下：

1、 SIP TP 管理从接入模块或者承载收到 SIP 消息，SIP TP 管理进程调用 PPLAT_SIP_TPMNG_DB_getTpIDForUpMsg 获取 TP 处理进程标识，并将消息发送至 TP 处理进程。

2、 SIP TP 管理进程从 SIP TR/TU 进程接收到下行消息，SIP TP 管理进程调用 PPLAT_SIP_TPMNG_DB_getTpIDForDownMsg 获取 TP 处理进程标识，并将消息发送至 TP 处理进程。

3、 SIP TP 管理进程收到从 SIP TP 发送过来的下行消息，SIP TP 管理进程通过承载接口将消息发送出去。

6.1.9 SIP TU 管理进程实现原理

SIP TU 管理进程主要负责上下行消息的分发，其分发的策略由具体项目决定。在某些实现情况下，上下行消息还可以不经过 SIP TU 管理进程，比如 SSS2.0 的分发实现。

SIP TU 管理进程的基本分发场景如下：

1、 上行初始请求的分发，SIP TR/TU 在从 TP 收到上行初始请求时调用 PPLAT_SIP_TUMNG_DB_getTuIDForUpMsg 获取 TR/TU 处理进程标识，并将上行消息发送至 TU 处理进程

2、 下行初始请求的分发，SIP TR/TU 在从应用收到下行初始请求时调用 PPLAT_SIP_TUMNG_DB_getTuIDForDownMsg 获取 TR/TU 处理进程消息，并将下行消息发送至 TR/TU 处理进程。

3、 下行初始 Forking 请求的分发，SIP TR/TU 在从应用收到下行初始 Forking 请求时调用 PPLAT_SIP_TUMNG_DB_getTuIDForForkingDownMsg 获取 TR/TU 处理进程标识，并将消息发送至 TR/TU 处理进程。

6.1.10 主备倒换

TU 管理进程目前不存在数据区实例及其他类似数据，因此不存在数据同步的问题，在主备倒换的时候只需要进行状态切换即可。

TP 管理进程的数据区仅有链路数据区、SOCKET 数据区、IPSecSA 数据区，其倒换处理和 TP 处理进程类似，这里也不单独描述。其倒换处理可参见《NGN PPLAT V1.00.10_SIP 协议 TP 模块（TP）详细设计说明.doc》

SIP TP 处理进程和 SIP TR/TU 处理进程主备倒换没有变更。

6.1.11 添加 record-route 与修改 record-route

注：修订记录编号 0001

原有 SIP V1.00.10 版本中，在添加本地 record-route 时，对于添加域名形式的，只添加了域名，而在有些项目中要求指定端口号，可以通过在 PPLATAPP 中增加项目开关项，来控制此场景下是否添加指定端口到 record-route 中，如果指定添加，则将本端的端口号添加到 record-route 中。

响应下发时，在入呼侧需要修改 record-route，在 SIP V1.00.10 版本中，修改了 zte-did 的内容，如果 record-route 中添加了端口号，可能需要修改其中的端口号。可以通过在

PPLATAPP 中增加项目开关项，来控制此场景下是否修改 record-route 中的端口号，如果指定修改，则将入呼侧数据区的本端端口号修正到 record-route 中。

此开关项定义如下，是否打开有项目进行选择。

```
/* 是否在 RECORD-ROUTE 中添加 PORT 参数 */
#define PPLAT_SIP_ADD_RECORDROUTE_PORT                    TRUE

/* 是否修改 RECORD-ROUTE 中 IP-PORT 参数 */
 #define PPLAT_SIP_MODIFY_RECORDROUTE_PORT                TRUE
```

6.1.12 Record-route 优化

注：修订记录编号 0003

一、问题提出的背景以及处理原则

SIP 平台此修改是为了兼容 SSS 中移测试的需求，对 UA 情况下 record-route 的填写处理进行优化，SIP 模块对于 Record-Route 的处理原则如下：

1、 当 SIP 接收初始会话请求时（作为 UAS），需要创建路由集；（补充：sss1.6 的实现中，不论该初始 Invite 请求是否携带 Record-Route 头部，sss 回复的响应中均将自身 ip 地址加入 R-R 中，便于后续请求路由以及分发；但是由于 pss2.0 不存在分发的问题，因此 sip 平台的处理维持原来的处理不变，即回复的响应中不将自身 ip 地址加入 R-R 中）

2、 当 SIP 发送初始会话请求时（作为 UAC），携带 R-R 头部且置头部为自身 ip 地址；接收 Invite 的响应（18x/2xx）时，需要创建路由集；

3、 当 SIP 接收后续请求时（SIP 作 UAC 或 UAS），若该后续请求中未携带 R-R 头部，则 SIP 回复的该对话内请求对应响应中不携带 R-R 头部；若该后续请求中携带了 R-R 头部，则 SIP 回复的该对话内请求对应响应中携带 R-R 头部，且 R-R 与初始 Invite 事务协商的保持一致而非该对话内请求中的 R-R 头部；

4、 当 SIP 发送后续请求时（SIP 作 UAC 或 UAS），不携带 R-R 头部，携带 Route 头部指定路由（Route 头部从初始 Invite 事务创建的路由集获得）；接收后续请求的响应不再更新路由集；

二、更改方案

（1） TU收到TR的上行请求时，检测对话内请求是否包含R-R头部，如果有的话，则在添加事务列表中的同时，就将blHasRecordRoute标志置为TRUE，否则置为FALSE

```
typedef struct tagSIP_TU_TRANSACTION_ITEM_T
{
    BOOL8                blIsUsed;
    BYTE                 bListIndex;
    BOOL8                blIsClient;
    BOOL8                blHasRecordRoute;   //新增加，表明对话内请求是否添加
                                                R-R
    BYTE                 bPeerTransIndex;
    WORD16                wMethod;
    BYTE                 bPad;
    WORD32                dwCSeq;
    PPLAT_SIP_SAP_ID_T   tTransID;
```

}PPLAT_SIP_PACK SIP_TU_TRANSACTION_ITEM_T;

（2） 确认所有回送响应的场景，在生成的码流添加R-R头部之前，判断是否是初始请求，再决定是否添加

2.1 如果是初始请求，则直接添加R-R

2.2 如果不是初始请求，则根据生成的消息以及数据区查找事务列表，取出对应事务的blHasRecordRoute标志，如果为TRUE，则添加R-R，否则不添加

当前判断是否初始请求以及查找事务列表都是根据ptleg(数据区)以及ptmsg（新生成的消息）来进行判断，比如要生成的update的200，则依据产生的这个200的消息中的方法名以及cseq来判断是否为初始请求。

异常情况的考虑：

如果对话内请求包含错误的 R-R 头部的情况，考虑的处理为不检查 R-R 头部的正确性，只检测是否包含此头部，如果有的话，则回送的响应携带正确的路由集中保存的 R-R。

6.2 进程设计

V1.01 新增两个进程为 TU 管理进程和 TP 管理进程。V1.0 版本原有的 TU 进程和 TP 进程不变，其详细内容可参见 V1.0 版本相关设计。

6.3 状态机设计

6.3.1 TU 管理进程状态机设计

此处描述 TUMNG 进程状态机

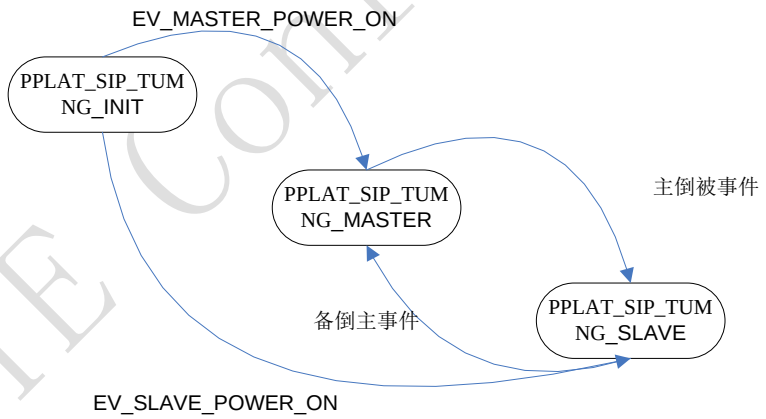


图 6.1 TU 管理进程状态

各状态含义见下表

表 6.1

状态名称	所属状态	状态说明
PPLAT_SIP_TUMNG_INIT	无	未上电
PPLAT_SIP_TUMNG_MASTER	无	主用状态
PPLAT_SIP_TUMNG_SLAVE	无	备用状态

各状态迁移关系见下表

表 6.2

源状态	触发事件	监护条件	目标状态
PPLAT_SIP_TUMNG_INIT	EV_MASTER_POWER_ON	无	PPLAT_SIP_TUMNG_MASTER
	EV_SLAVE_POWER_ON	无	PPLAT_SIP_TUMNG_SLAVE
PPLAT_SIP_TUMNG_MASTER	主倒被事件	无	PPLAT_SIP_TUMNG_SLAVE
PPLAT_SIP_TUMNG_SLAVE	被倒主事件	无	PPLAT_SIP_TUMNG_MASTER

6.3.2 TP 管理进程状态机设计

此处描述 TPMNG 进程状态机

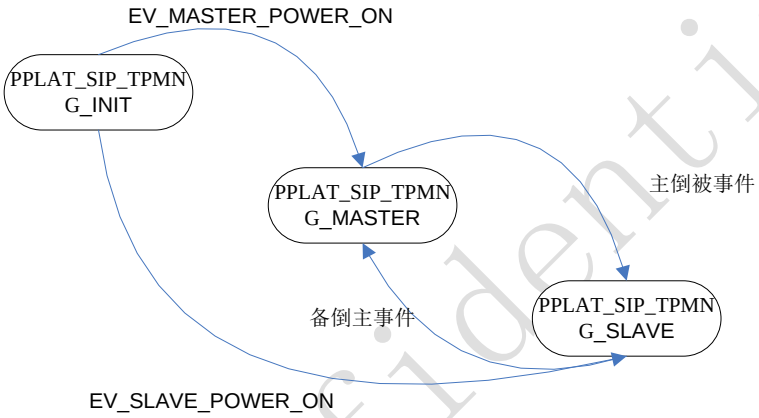


图 6.2 TP 管理进程状态

各状态含义见下表

表 6.3

状态名称	所属状态	状态说明
PPLAT_SIP_TPMNG_INIT	无	未上电
PPLAT_SIP_TPMNG_MASTER	无	主用状态
PPLAT_SIP_TPMNG_SLAVE	无	备用状态

各状态迁移关系见下表

表 6.4

源状态	触发事件	监护条件	目标状态
PPLAT_SIP_TPMNG_INIT	EV_MASTER_POWER_ON	无	PPLAT_SIP_TPMNG_MASTER
	EV_SLAVE_POWER_ON	无	PPLAT_SIP_TPMNG_SLAVE
PPLAT_SIP_TPMNG_MASTER	主倒被事件	无	PPLAT_SIP_TPMNG_SLAVE
PPLAT_SIP_TPMNG_SLAVE	被倒主事件	无	PPLAT_SIP_TPMNG_MASTER

6.4 关键数据设计

无

6.5 关键算法设计

无

6.6 错误定位手段

6.6.1 流程跟踪

TP 处理进程的流程跟踪需要增加如下几个跟踪点，

1. 收到 TP 管理进程发送的上行消息
2. 向 TP 管理进程发送下行消息。

TU 管理进程的流程跟踪需要跟中如下几个跟踪点，

1. 收到 TP 上行消息
2. 向 TU 处理进程发送上行消息
3. 向 TU 处理进程发送下行消息
4、收到下行消息

TP 管理进程不在流程内，不进行流程跟踪。

6.6.2 失败观察

TP 管理进程、TU 管理进程在发生失败的地方均上报失败观察。

6.6.3 统计

TP 处理的统计点增加从 TP 管理进程接收和向管理进程发送消息。

TP 管理进程统计从底层 BRS 或者 SLB 接收的消息，向 BRS 或者 SLB 发送的消息。

TU 管理进程统计上下行消息的收发。

6.6.4 信令跟踪

TP 管理进程、TU 管理进程、TP 处理进程、TP 管理进程均增加对于跟踪类任务设置与取消事件的处理，以适应 SIP 平台自行维护任务队列的场景。

6.6.5 告警通知

TP 的告警通知有：获取数据区配置失败、获取相关进程 PID 失败、主备倒换初始失败、申请数据区失败、向主备模块注册备份数据失败、SOCKET 初始化失败超过 20 次、链路初始化失败超过 20 次。当 TP 模块发生上述失败时，向网管前台 AGENT 发送告警消息。系统重新上电或 SOCKET 初始化、链路初始化成功后，网管感知后将自动恢复（清除）所有告警。系统上电失败对应的告警级别为 PPLAT_SIP_OMM_ALARM_HIGH 告警。

当主备板上电时，若从 DB 接口获得 TP 数据区个数失败时，调用上电失败告警函数函数发起告警。将告警模块设置为 PPLAT_SIP_MODULE_TP；将错误类型设置为 PPLAT_SIP_OMM_ALARM_CFG；并在附加信息中填写错误信息的描述。

当主备板上电时，获取相关进程的 PID 失败时，调用上电失败告警函数发起告警。将告警模块设置为 PPLAT_SIP_MODULE_TP；将错误类型设置为 PPLAT_SIP_OMM_ALARM_MEM；并在附加信息中填写错误信息的描述。

当主备板上电时，主备倒换初始化失败时，调用上电失败告警函数发起告警。将告警模块设置为 PPLAT_SIP_MODULE_TP；将错误类型设置为 PPLAT_SIP_OMM_ALARM_CFG；并在附加信息中填写错误信息的描述。

当主备板上电时，TP 模块向主备模块注册数据区备份失败时，调用上电失败告警函数发起告警。将告警模块设置为 PPLAT_SIP_MODULE_TP；将错误类型设置为 PPLAT_SIP_OMM_ALARM_CFG；并在附加信息中填写错误信息的描述。

当主备板上电时，TP 模块申请进程数据区失败时，调用上电失败告警函数发起告警。

将告警模块设置为 PPLAT_SIP_MODULE_TP；将错误类型设置为 PPLAT_SIP_OMM_ALARM_MEM；并在附加信息中填写错误信息的描述。

当主备板上电初始化 SOCKET 或链路数据区，或 SOCKET 和链路初始化定时器超时，如果 SOCKET 或链路初始化次数超过 20 次，分别调用 SOCKET 初始化、链路初始化失败告警函数发起告警。将告警模块设置为 PPLAT_SIP_MODULE_TP；将错误类型设置为 PPLAT_SIP_OMM_ALARM_MEM；并在附加信息中填写错误信息的描述。

当 SOCKET 或链路状态从不可用状态变换到正常状态时，且之前发送过告警，则调用函数 PPLAT_SIP_OMM_notify 发送通知消息至网管 AGENT。此外，链路人工闭塞 / 解闭塞时，也需要发送通知消息。

6.6.6 数据区观察

数据区的观察功能当前主要有五类：数据区个数统计查询，按时间状态数据区查询，指定单个数据区内容查询，指定数据区删除，指定索引范围内的数据区批量删除。

当 TP 进程收到数据区统计查询或数据区实例内容查询请求，调用数据区统计查询接口上报统计信息，在该接口函数内根据查询的数据区类型以及其它条件，构造报文并上报给网管。

6.6.7 日志打印

SIP 协议栈将打印等级分为以下五项：

PPLAT_SIP_OMM_PRINT_LOWEST：简单说明性打印

PPLAT_SIP_OMM_PRINT_LOW：正常打印，该打印指明关键流程方向

PPLAT_SIP_OMM_PRINT_NORMAL：出现了错误，但不影响流程

PPLAT_SIP_OMM_PRINT_HIGH：出现了错误可能会影响流程

PPLAT_SIP_OMM_PRINT_HIGHEST：出现了严重的错误，会导致流程中止或流程变更

当 TP 在进行日志打印时，可通过调用 PPLAT_SIP_OMM_print 函数来实现。将其中的 bModuleType 设置为事务层的模块 PPLAT_SIP_MODULE_TP，再根据不同的情况设置打印等级，打印等级说明如上，最后设置打印内容。

在机架环境上使用此函数时，首先判断打印开关是否打开，若否就不打印日志。若开关已打开，判断输入等级和模块是否需要打印，若否就不打印日志；否则就打印日志。在 VC 版本上，进行日志的打印过程与机架相似，但是在 VC 上打印时，若本地日志文件不存在会创建此本地日志文件。

TP 层进行日志打印时，将模块置为 PPLAT_SIP_MODULE_TP，在为开发调试定位问题需要的地方，都可加上日志。

6.7 兼容性设计

在系统方案中，配置及网管相关接口是可重载的。不同层面的分发接口也是可重载的。另外为了兼容 V1.0 版本，采用编译宏区分不同平台。

7 数据描述

7.1 数据结构说明

无

7.2 全局变量和常量说明

无

7.3 数据库说明

无

8 界面设计

无。

9 类设计

无。

10 函数设计

10.1 P_SIP_ALLOCATOR_create

10.1.1 概述

函数名称	P_SIP_ALLOCATOR_create	类型（函数/宏）	功能函数
功能详细说明	内存分配器创建接口，由外部指示内存分配器管理的内存资源是由外部传入还是由内存分配器自行申请。在该接口中队内存分配器结构进行初始化，建立对内存资源的管理队列。当内存由外部传入时，如果传入的内存不足，内存分配器创建失败；当内存由内存分配器自行申请时，如果申请失败，内存分配器创建失败		
性能			

序号	分类	使用的全局及静态变量
1	全局	无
2	静态	无

资源限制

函数声明
BOOL8 P_SIP_ALLOCATOR_create(IN PPLAT_SIP_ALLOCATOR_CREATE_IN_T* ptCreateIn, OUT PPLAT_SIP_ALLOCATOR_T* ptAllocator);

序号	参数类型	返回值说明
1	BOOL8	成功返回 TRUE；失败返回 FALSE
序号	参数名	输入参数说明
1	ptCreateIn	创建内存分配器入参结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》
序号	参数名	输出参数说明
1	ptAllocator	内存分配器结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》

10.1.2 处理逻辑

BOOL8 P_SIP_ALLOCATOR_create(IN PPLAT_SIP_ALLOCATOR_CREATE_IN_T* ptCreateIn, OUT PPLAT_SIP_ALLOCATOR_T* ptAllocator)

{

WORD32 dwTotoalSize=0;

BYTE * pbEnd=NULL;

空指针保护;

dwTotoalSize=计算所需要的内存大小;

if(ptCreateIn-> blIsNeedMalloc)

```
{
    ptAllocator-> pbBuf= P_SIP_malloc(dwTotoalSize);
    if(ptAllocator-> pbBuf==NULL)
    {
        打印错误日志;
        return false;
    }
    else
    {
        ptAllocator-> blIsMallocBySelt=TRUE;
        pbEnd= ptAllocator-> pbBuf+ dwTotoalSize;
    }
}
else
{
    if(ptCreateIn-> dwBufLen< dwTotoalSize)
    {
        打印错误日志;
        return false;
    }
    else
    {
        ptAllocator->pbBuf= ptCreateIn->pbBuf;
        ptAllocator->blIsMallocBySelt=False;
        pbEnd= ptAllocator->pbBuf+ ptCreateIn->dwBufLen;
    }
}
/* 初始化 1K 内存池 */
if(!SIP_ALLOCATOR_initMemPool(ptAllocator->pbBuf, pbEnd-ptAllocator->pbBuf, 1024, ptCreateIn-> dw1kNum, &ptAllocator-> t1KMemPool))
{
    如果内存是自行申请的，则释放内存;
    打印日志;
    return false;
}
/* 初始化 2K 内存池 */
if(!SIP_ALLOCATOR_initMemPool(ptAllocator->t1KmemPool.pbMemBufEnd, pbEnd-ptAllocator->t1KmemPool.pbMemBufEnd, 2048, ptCreateIn->dw2kNum, &ptAllocator->t2KMemPool))
{
    如果内存是自行申请的，则释放内存;
    打印日志;
    return false;
}
```

```
        /* 初始化 4K 内存池 */
        if(!SIP_ALLOCATOR_initMemPool(ptAllocator->t2KmemPool.pbMemBufEnd, pbEnd-
ptAllocator-> t2KmemPool.pbMemBufEnd, 4*1024, ptCreateIn-> dw4kNum, &ptAllocator-> t4KMemPool))
        {
            如果内存是自行申请的，则释放内存;
            打印日志;
            return false;
        }

        /* 初始化 8K 内存池 */
        if(!SIP_ALLOCATOR_initMemPool(ptAllocator->t4KmemPool.pbMemBufEnd, pbEnd-
ptAllocator-> t4KmemPool.pbMemBufEnd, 8*1024, ptCreateIn-> dw8kNum, &ptAllocator-> t8KMemPool))
        {
            如果内存是自行申请的，则释放内存;
            打印日志;
            return false;
        }

        /* 初始化 16K 内存池 */
        if(!SIP_ALLOCATOR_initMemPool(ptAllocator->t8KmemPool.pbMemBufEnd,pbEnd-
ptAllocator-> t8KmemPool.pbMemBufEnd, 16*1024, ptCreateIn-> dw16kNum, &ptAllocator-> t16KMemPool))
        {
            如果内存是自行申请的，则释放内存;
            打印日志;
            return false;
        }

        /* 初始化 32K 内存池 */

if(!SIP_ALLOCATOR_initMemPool(ptAllocator->t16KmemPool.pbMemBufEnd,pbEnd-ptAllocator->t16Kmem
Pool.pbMemBufEnd, 32*1024, ptCreateIn-> dw32kNum, &ptAllocator-> t32KMemPool))
        {
            如果内存是自行申请的，则释放内存;
            打印日志;
            return false;
        }

        ptAllocator-> blIsInit=TRUE;
        return TRUE;
    }
```

10.2 SIP_ALLOCATOR_initMemPool

10.2.1 概述

函数名称	SIP_ALLOCATOR_initMemPool	类型（函数/宏）	功能函数

功能详细说明		使用一块内存对对内存池进行初始化。在内存分配中，在内存的前部保存内存篇管理节点数组，在内存片的后部保存内存片。
性能		
序号	分类	使用的全局及静态变量
1	全局	无
2	静态	无
资源限制		
函数声明		
BOOL8 SIP_ALLOCATOR_initMemPool (IN BYTE* pbBuf,IN WORD32 dwBuflen,IN WORD32 dwNodeSize,IN WORD32 dwNodeNum,OUT PPLAT_SIP_ALLOCATOR_MEMPOOL_T ptMempool);		
序号	参数类型	返回值说明
1	BOOL8	成功返回 TRUE；失败返回 FALSE
序号	参数名	输入参数说明
1	pbBuf	用来初始化的缓冲区指针
2	dwBuflen	缓冲区长度
3	dwNodeSize	内存片大小
4	dwNodeNum	内存篇个数
5		
序号	参数名	输出参数说明
1	ptMempool	内存池管理结构指针

10.2.2 处理逻辑

```
BOOL8 SIP_ALLOCATOR_initMemPool (IN BYTE* pbBuf,IN WORD32 dwBuflen,IN WORD32 dwNodeSize,IN WORD32 dwNodeNum,OUT PPLAT_SIP_ALLOCATOR_MEMPOOL_T ptMempool)
{
    DWORD32 i;
    PPLAT_SIP_ALLOCATOR_MEMNODE_T* ptTemNode ;
    空指针保护 ;
    /* 空间大小检查 */
    if(dwBuflen<((sizeof(PPLAT_SIP_ALLOCATOR_MEMNODE_T)+ dwNodeSize)* dwNodeNum))
    {
        打印日志;
        return false;
    }
    P_SIP_memset(ptMempool,0,sizeof(PPLAT_SIP_ALLOCATOR_MEMPOOL_T));
    ptMempool-> dwNodeSize= dwNodeSize;
    ptMempool-> wTotalNum= dwNodeNum;
    ptMempool-> wUsedNum=0;
    ptMempool-> wIdleNum= dwNodeNum;
    ptMempool-> dwIdleHeadIndex=0;
    ptMempool-> ptMemNode= (PPLAT_SIP_ALLOCATOR_MEMNODE_T*) pbBuf ;
    ptMempool-> pbMemBufBegin= pbBuf +<((sizeof(PPLAT_SIP_ALLOCATOR_MEMNODE_T)* dwNodeNum ;
    ptMempool-> pbMemBufEnd = pbBuf ((sizeof(PPLAT_SIP_ALLOCATOR_MEMNODE_T)+ dwNodeSize)* dwNodeNum)) ;
```

```
    for(i=0 ;i< dwNodeNum ;i++)
    {
        ptTemNode = ptMempool-> ptMemNode+i;
        ptTemNode -> blIsUsed=FALSE;
        ptTemNode -> dwPstIndex=i+1;
    }
    ptTemNode -> dwPstIndex=0xffff ; /* 表示没有后继节点 */
    return true ;
}
```

10.3 P_SIP_ALLOCATOR_destroy

10.3.1 概述

函数名称	P_SIP_ALLOCATOR_destroy	类型（函数/宏）	功能函数
功能详细说明	内存分配器销毁接口，如果内存分配器管理的内资源由内存分配器自身申请的，则在本接口中释放内存分配器管理的内存资源；如果内存分配器管理的内资源由外部传入，则在本接口中不进行释放。		
性能			

序号	分类	使用的全局及静态变量
1	全局	无
2	静态	无

资源限制

函数声明
VOID P_SIP_ALLOCATOR_destroy (IN PPLAT_SIP_ALLOCATOR_T* ptAllocator);

序号	参数类型	返回值说明
1	无	

序号	参数名	输入参数说明
1	ptAllocator	内存分配器结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》

序号	参数名	输出参数说明

10.3.2 处理逻辑

```
VOID P_SIP_ALLOCATOR_destroy (IN PPLAT_SIP_ALLOCATOR_T* ptAllocator)
{
    空指针保护;
    /* 如果没有初始化或者内存不是由内存分配器申请的，则不进行释放 */
    if(ptAllocator-> blIsInit==FALSE ||  ptAllocator-> blIsMallocBySelt==FALSE  )
    {
        return ;
    }
    else /* 如果是内存分配器申请的 */
    {
        P_SIP_free(ptAllocator);
```

```
        }
        ptAllocator-> blIsInit=FALSE;
        return;
    }
```

10.4 P_SIP_ALLOCATOR_allocate

10.4.1 概述

函数名称	P_SIP_ALLOCATOR_allocate	类型（函数/宏）	功能函数
功能详细说明	内存分配器分配函数，从内存分配器管理的内存池中分配需要大小的内存，分配成功返回指向相应内存的指针，失败返回 NULL。		
性能			

序号	分类	使用的全局及静态变量
1	全局	无
2	静态	无

资源限制

函数声明
VOID* P_SIP_ALLOCATOR_allocate (IN PPLAT_SIP_ALLOCATOR_T *ptAllocate,IN WORD32 dwSize);

序号	参数类型	返回值说明
1	VOID*	成功，为分配的内存地址；失败，为 NULL；
序号	参数名	输入参数说明
1	ptAllocate	内存分配器结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》
2	dwSize	需要分配的内存大小
序号	参数名	输出参数说明
1		

10.4.2 处理逻辑

```
VOID* P_SIP_ALLOCATOR_allocate (IN PPLAT_SIP_ALLOCATOR_T *ptAllocate,IN WORD32 dwSize)
{
    VOID* pvOut=NULL;
    空指针保护;
    /* 内存分配器有效性检查 */
    if(!ptAllocate-> blIsInit)
    {
        打印错误日志;
        return NULL;
    }
    pvOut= SIP_ALLOCATOR_allocateFromMempool(&ptAllocate->t1KMemPool,dwSize);
    if(pvOut!=NULL)
    {
        return pvOut;
    }
    pvOut= SIP_ALLOCATOR_allocateFromMempool(&ptAllocate->t2KMemPool,dwSize);
    if(pvOut!=NULL)
```

```
    {
        return pvOut;
    }
    pvOut= SIP_ALLOCATOR_allocateFromMempool(&ptAllocate->t4KMemPool,dwSize);
    if(pvOut!=NULL)
    {
        return pvOut;
    }
    pvOut= SIP_ALLOCATOR_allocateFromMempool(&ptAllocate->t8KMemPool,dwSize);
    if(pvOut!=NULL)
    {
        return pvOut;
    }
    pvOut= SIP_ALLOCATOR_allocateFromMempool(&ptAllocate->t16KMemPool,dwSize);
    if(pvOut!=NULL)
    {
        return pvOut;
    }
    pvOut= SIP_ALLOCATOR_allocateFromMempool(&ptAllocate->t32KMemPool,dwSize);
    if(pvOut!=NULL)
    {
        return pvOut;
    }
    return NULL;
}
```

10.5 SIP_ALLOCATOR_allocateFromMempool

10.5.1 概述

函数名称	SIP_ALLOCATOR_allocateFromMempool	类型（函数/宏）	功能函数
功能详细说明	从内存池中分配固定大小的内存片，如果需要分配的内存超过内存池中固定内存片的大小或者内存池中没有空闲内存片，则分配失败，返回空指针。		
性能			

序号	分类	使用的全局及静态变量
1	全局	无
2	静态	无

资源限制

函数声明
VOID* SIP_ALLOCATOR_allocateFromMempool(IN PPLAT_SIP_ALLOCATOR_MEMPOOL_T *ptMemPool,IN WORD32 dwSize);

序号	参数类型	返回值说明
1	VOID*	成功，为分配的内存地址；失败，为 NULL；
序号	参数名	输入参数说明
1	ptMemPool	内存池结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》

2	dwSize	需要分配的内存大小
序号	参数名	输出参数说明
1		

10.5.2 处理逻辑

VOID* SIP_ALLOCATOR_allocateFromMempool(IN PPLAT_SIP_ALLOCATOR_MEMPOOL_T *ptMemPool,IN WORD32 dwSize)

{

空指针保护;

if(ptMemPool->wIdleNum==0|| ptMemPool-> dwNodeSize< dwSize)

{

return NULL;

}

将内存池内的首空闲内存片分配出去;

将原首空闲内存片分配的后续内存片作为新的首空闲内存片;

将空闲内存片个数减一，占用内存片个数加一;

}

10.6 P_SIP_ALLOCATOR_deallocate

10.6.1 概述

函数名称	P_SIP_ALLOCATOR_deallocate	类型（函数/宏）	功能函数
功能详细说明	内存分配器内存释放函数，将从内存分配器中申请的内存释放，使对应的内存可以重新被分配。		
性能			
序号	分类	使用的全局及静态变量	
1	全局	无	
2	静态	无	
资源限制			
函数声明			
VOID P_SIP_ALLOCATOR_deallocate(IN PPLAT_SIP_ALLOCATOR_T *ptAllocator,IN VOID* ptr);			
序号	参数类型	返回值说明	
1	VOID	无	
序号	参数名	输入参数说明	
1	ptAllocator	内存分配器结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》	
2	ptr	指向需要分配的内存指针	
序号	参数名	输出参数说明	
1			
2			

10.6.2 处理逻辑

VOID P_SIP_ALLOCATOR_deallocate(IN PPLAT_SIP_ALLOCATOR_T *ptAllocator,IN VOID* ptr)

{

空指针保护;

if(ptAllocator-> blIsInit==FALSE)

```
        {
            return ;
        }
        if(ptr>=ptAllocator->t1KmemPool.pbMemBufBegin&&ptr<ptAllocator->t1KmemPool.
pbMemBufEnd)
        {
            SIP_ALLOCATOR_deallocateFromMemPool(&ptAllocator->t1KmemPool, ptr);
        }
        elseif(ptr>=ptAllocator->t2KmemPool.pbMemBufBegin&&ptr<ptAllocator->t2KmemPool.
pbMemBufEnd)
        {
            SIP_ALLOCATOR_deallocateFromMemPool(&ptAllocator->t2KmemPool, ptr);
        }
        elseif(ptr>=ptAllocator->t4KmemPool.pbMemBufBegin&&ptr<ptAllocator->t4KmemPool.
pbMemBufEnd)
        {
            SIP_ALLOCATOR_deallocateFromMemPool(&ptAllocator->t4KmemPool, ptr);
        }
        elseif(ptr>=ptAllocator->t8KmemPool.pbMemBufBegin&&ptr<ptAllocator->t8KmemPool.
pbMemBufEnd)
        {
            SIP_ALLOCATOR_deallocateFromMemPool(&ptAllocator->t8KmemPool, ptr);
        }
        elseif(ptr>=ptAllocator->t16KmemPool.pbMemBufBegin&&ptr<ptAllocator->t16KmemPool.
pbMemBufEnd)
        {
            SIP_ALLOCATOR_deallocateFromMemPool(&ptAllocator->t16KmemPool, ptr);
        }
        elseif(ptr>=ptAllocator->t32KmemPool.pbMemBufBegin&&ptr<ptAllocator->t32KmemPool.
pbMemBufEnd)
        {
            SIP_ALLOCATOR_deallocateFromMemPool(&ptAllocator->t32KmemPool, ptr);
        }
        else
        {
            打印日志;
        }
        reurn;
    }
```

10.7 SIP_ALLOCATOR_deallocateFromMemPool

10.7.1 概述

函数名称	SIP_ALLOCATOR_deallocateFromMemPool	类型（函数/宏）	功能函数

功能详细说明		向内存池中释放内存片，如果释放内存片的指针不在内存池内，或者不在内存池内存片边界点上，则释放失败。
性能		
序号	分类	使用的全局及静态变量
1	全局	无
2	静态	无
资源限制		
函数声明		
BOOL8 SIP_ALLOCATOR_deallocateFromMemPool(IN PPLAT_SIP_ALLOCATOR_MEMPOOL_T *ptMemPool,IN VOID* ptr);		
序号	参数类型	返回值说明
1	BOOL8	TRUE—释放成功; FALSE—释放失败;
序号	参数名	输入参数说明
1	ptAllocator	内存分配器结构指针,结构定义参见《NGN PPLAT SIP V1.01.10_接口说明书.doc》
2	ptr	指向需要分配的内存指针
序号	参数名	输出参数说明
1		
2		

10.7.2 处理逻辑

BOOL8 SIP_ALLOCATOR_deallocateFromMemPool(IN PPLAT_SIP_ALLOCATOR_MEMPOOL_T *ptMemPool,IN VOID* ptr)

{

WORD32 i=0;

空指针保护;

if(ptr< ptMemPool->pbMemBufBegin|| ptr>=ptMemPool->pbMemBufEnd)

{

return false;

}

if(!((ptr- ptMemPool->pbMemBufBegin)% ptMemPool-> dwNodeSize))

{

打印错误日志;

return false;

}

i=(ptr- ptMemPool->pbMemBufBegin)/ ptMemPool-> dwNodeSize;

将 i 对应的内存片节点放在空闲内存片队列首;

return True;

}

10.8 PPLAT_SIP_TUMNG_MainEntry

10.8.1 概述

名称	PPLAT_SIP_TUMNG_MainEntry	类型	主函数
功能	TU 管理进程入口函数		

性能		无
序号	分类	被调用函数
1	外	无
2	内	
序号	分类	使用的公共数据
1	外	无
2	内	无
资源限制		
函数声明		
void PPLAT_SIP_TUMNG_MainEntry (IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP);		
序号	分类	返回值说明
1	无	无
序号	分类	输入输出数据说明
1	wState	进程状态 PPLAT_SIP_TUMNG _INIT 0 PPLAT_SIP_TUMNG _MASTER 1 PPLAT_SIP_TUMNG _SLAVE 2
2	wMsgId	事件号
3	pMsgPara	事件指针
4	pVarP	变量指针

10.8.2 处理逻辑

```
void PPLAT_SIP_TUMNG_MainEntry(WORD16 wState, WORD16 wMsgId, LPVOID pMsgPara,
LPVOID pVarP)
{
    WORD32 dwResult=0;
    switch ( wState ) {
    /* 进程初始化 */
    case PPLAT_SIP_TUMNG_INIT:
        switch(wMsgId)
        {
        case EV_MASTER_POWER_ON:
            dwResult=SIP_TUMNG_init(TRUE);
            if(dwResult==成功)
            {
                切换到主板工作态;
            }
            else
            {
                打印错误日志并发送失败观察;
            }
            break;
        case EV_SLAVE_POWER_ON:
            dwResult=SIP_TUMNG_init(FALSE);
            if(dwResult==成功)
```

```
        {
            切换到备板工作态;
        }
        else
        {
            打印错误日志并发送失败观察;
        }
        break;
    default:
        break;
    }
    Break;
/* 主板处理 */
case PPLAT_SIP_TUMNG_MASTER:
    /* 定时器超时事件处理 */
    if( wMsgId is timer)
    {
        dwResult=SIP_TUMNG_Proc_Timer(wState, wMsgId, pMsgPara, pVarp);
        Break;
    }
    Switch(wMsgId)
    {
    /* 主倒备 */
    case EV_PPLAT_SIP_MASTER_TO_SLAVE:
        dwResult=SIP_TUMNG_master2slave();
        切换到备板状态;
        break;
    /* 上行消息处理 */;
    case EV_PPLAT_SIP_TPUI_IND:
        dwResult=SIP_TUMNG_onInd(pVarp);
        break;
    /* 下行Forking请求处理 */
    case EV_PPLAT_SIP_API_FORKING_REQ:
        dwResult=SIP_TUMNG_onForkingReq(pVarp);
        break;
    /*下行初始请求处理*/
    case EV_PPLAT_SIP_API_INIT_REQUEST_REQ:
        dwResult=SIP_TUMNG_onInitReq(pVarp);
        break;

    default:
        break;
    }
    break;
```

```
        /* 备板处理 */
        case PPLAT_SIP_TUMNG_SLAVE:
             switch(wMsgId)
            {
            case  EV_PPLAT_SIP_SLAVE_TO_MASTER:
                dwResult=SIP_TUMNG_slave2Master();
                切换到主板状态;
                break;
            default:
               break;
            }
             break;
        default:
             break;
        }   /*end of switch*/
        if(dwResult不为成功)
        {
            打印错误日志;
        }
        PPLAT_SIP_OSS_SetDefaultNextState_wrap();
    } /* end of PPLAT_SIP_MainEntry */
```

10.9 SIP_TUMNG_init

10.9.1 概述

名称		SIP_TUMNG_init	类型	主函数
功能		TU 管理进程入口函数，TU 管理进程初始化函数。在目前的实现中 TU 管理进程没有什么资源需要申请的，因此在该函数中仅进行 g_tSIP_TUMNG_ParserConfig 的初始化。		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	g_tSIP_TUMNG_ParserConfig		
资源限制				
函数声明				
WORD32 SIP_TUMNG_Proc_Timer(BOOL8 blIsMaster);				
序号	分类	返回值说明		
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败		
序号	分类	输入输出数据说明		
1	blIsMaster	是否是主板		

10.9.2 处理逻辑

WORD32 SIP_TUMNG_init(BOOL8 blIsMaster)

{

初始化 g_tSIP_TUMNG_ParserConfig;

打印日志;

return PPLAT_SIP_SUCCESS;

}

10.10 SIP_TUMNG_Proc_Timer

10.10.1 概述

名称		SIP_TUMNG_init	类型	主函数
功能		TU 管理进程定时器超时事件处理函数，TU 管理进程目前没有设置定时器，因此在该函数中什么都不做。		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
WORD32 SIP_TUMNG_Proc_Timer(IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP);				
序号	分类	返回值说明		
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败		
序号	分类	输入输出数据说明		
1	wState	进程状态 PPLAT_SIP_TUMNG_STACK_INIT 0 PPLAT_SIP_TUMNG _STACK_MASTER 1 PPLAT_SIP_TUMNG _STACK_SLAVE 2		
2	wMsgId	事件号		
3	pMsgPara	事件指针		
4	pVarP	变量指针		

10.10.2 处理逻辑

WORD32 SIP_TUMNG_Proc_Timer(IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP)

{

/* 该函数目前什么都不需要做 */

打印日志;

return PPLAT_SIP_SUCCESS;

}

10.11 SIP_TUMNG_master2slave

10.11.1 概述

名称		SIP_TUMNG_master2slave	类型	主函数
功能		TU 管理进程主倒备函数，TU 管理进程目前没有申请什么资源，因此该函数中什么都不需要做		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
WORD32 SIP_TUMNG_master2slave();				
序号	分类	返回值说明		
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败		
序号	分类	输入输出数据说明		

10.11.2 处理逻辑

```
WORD32 SIP_TUMNG_master2slave()
{
    打印日志;
    return PPLAT_SIP_SUCCESS;
}
```

10.12 SIP_TUMNG_slave2Master

10.12.1 概述

名称		SIP_TUMNG_slave2Master	类型	主函数
功能		TU 管理进程备倒主函数，TU 管理进程目前没有申请什么资源，因此该函数中什么都不需要做		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				

函数声明		
WORD32 SIP_TUMNG_slave2Master();		
序号	分类	返回值说明
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败
序号	分类	输入输出数据说明

10.12.2 处理逻辑

WORD32 SIP_TUMNG_slave2Master()

{

打印日志;

return PPLAT_SIP_SUCCESS;

}

10.13 SIP_TUMNG_onInd

10.13.1 概述

名称	SIP_TUMNG_onInd	类型	主函数
功能	TU 管理进程收到 TP 上行消息的处理函数，该函数不进行具体的消息处理，调用分发函数确定 TU 处理进程后将消息转发给 TU 处理进程处理		
性能	无		
序号	分类	被调用函数	
1	外	无	
2	内		
序号	分类	使用的公共数据	
1	外	无	
2	内	无	
资源限制			
函数声明			
WORD32 SIP_TUMNG_onInd(LPVOID pVarp);			
序号	分类	返回值说明	
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败	
序号	分类	输入输出数据说明	
1	pVarp	入参 结构指针	

10.13.2 处理逻辑

WORD32 SIP_TUMNG_onInd(VOID pVarp);

{

```
PPLAT_SIP_PARSER_CONFIG_T *ptConfig=NULL;
PPLAT_SIP_TPUI_IND_T *ptInd=NULL;
PPLAT_SIP_APPID_T    tDstId;
WORD32 dwResult=0;

空指针保护;
ptInd=( PPLAT_SIP_TPUI_IND_T *) pVarp;
ptConfig=SIP_TUMNG_getParserConfig();
dwResult== PPLAT_SIP_TUMNG_DB_getTuIDForUpMsg(ptInd, ptConfig,&tDstId);

if(dwResult!=PPLAT_SIP_SUCCESS)
{
    打印日志;
    上报失败观察等;
    return dwResult;
}
发送 IND 消息到 TU 处理进程;
返回发送结果;
}
```

10.14 SIP_TUMNG_onForkingReq

10.14.1 概述

名称		SIP_TUMNG_onForkingReq	类型	主函数
功能		TU 管理进程收到下行 ForkingReq 消息的处理函数，该函数不进行具体的消息处理，调用分发函数确定 TU 处理进程后将消息转发给 TU 处理进程处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
WORD32 SIP_TUMNG_onForkingReq(LPVOID pVarp);				
序号	分类	返回值说明		
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败		
序号	分类	输入输出数据说明		
1	pVarp	入参 结构指针		

10.14.2 处理逻辑

```
WORD32 SIP_TUMNG_onForkingReq (VOID pVarp);
{
    PPLAT_SIP_PARSER_CONFIG_T *ptConfig=NULL;
    PPLAT_SIP_FORKING_REQ_T *ptReq=NULL;
    PPLAT_SIP_APPID_T    tDstId;
    WORD32 dwResult=0;

    空指针保护;
    ptReq =( PPLAT_SIP_FORKING_REQ_T *) pVarp;
    ptConfig=SIP_TUMNG_getParserConfig();
   dwResult== PPLAT_SIP_TUMNG_DB_getTuIDForForkingDownMsg (ptReq, ptConfig,&tDstId);

   if(dwResult!=PPLAT_SIP_SUCCESS)
   {
       打印日志;
       上报失败观察等;
       return dwResult;
   }
   发送 REQ 消息到 TU 处理进程;
   返回发送结果;
}
```

10.15 SIP_TUMNG_onInitReq

10.15.1 概述

名称		SIP_TUMNG_onInitReq	类型	主函数
功能		TU 管理进程收到下行初始请求消息的处理函数，该函数不进行具体的消息处理，调用分发函数确定 TU 处理进程后将消息转发给 TU 处理进程处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
WORD32 SIP_TUMNG_onInitReq(LPVOID pVarp);				
序号	分类	返回值说明		
1	无	PPLAT_SIP_SUCCESS---成功 其他，失败		
序号	分类	输入输出数据说明		
1	pVarp	入参 结构指针		

10.15.2 处理逻辑

```
WORD32 SIP_TUMNG_onInd(VOID pVarp);
{
    PPLAT_SIP_PARSER_CONFIG_T *ptConfig=NULL;
    PPLAT_SIP_INIT_REQUEST_REQ_T *ptReq=NULL;
    PPLAT_SIP_APPID_T    tDstId;
    WORD32 dwResult=0;

    空指针保护;
    ptReq =( PPLAT_SIP_INIT_REQUEST_REQ_T *) pVarp;
    ptConfig=SIP_TUMNG_getParserConfig();
   dwResult== PPLAT_SIP_TUMNG_DB_getTuIDForDownMsg (ptReq, ptConfig,&tDstId);

    if(dwResult!=PPLAT_SIP_SUCCESS)
    {
        打印日志;
        上报失败观察等;
        return dwResult;
    }
    发送 REQ 消息到 TU 处理进程;
    返回发送结果;
}
```

10.16 SIP_TUMNG_getParserConfig

10.16.1 概述

名称		SIP_TUMNG_getParserConfig	类型	主函数
功能		TU 管理进程获取编解码配置函数。		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	g_tSIP_TUMNG_ParserConfig		
资源限制				
函数声明				
PPLAT_SIP_PARSER_CONFIG_T* SIP_TUMNG_getParserConfig ();				
序号	分类	返回值说明		
1	无	编解码配置结构指针		
序号	分类	输入输出数据说明		
1				

10.16.2 处理逻辑

```
PPLAT_SIP_PARSER_CONFIG_T* SIP_TUMNG_getParserConfig ()
{
    return &g_tSIP_TUMNG_ParserConfig;
}
```

10.17 PPLAT_SIP_TPMNG_MainEntry

10.17.1 概述

名称		PPLAT_SIP_TPMNG_MainEntry	类型	主函数
功能		TP 管理进程入口函数		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
void PPLAT_SIP_TPMNG_MainEntry (IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP);				

序号	分类	返回值说明
1	无	无
序号	分类	输入输出数据说明
1	wState	进程状态 PPLAT_SIP_TPMNG_INIT 0 PPLAT_SIP_TPMNG_MASTER 1 PPLAT_SIP_TPMNG_SLAVE 2
2	wMsgId	事件号
3	pMsgPara	事件指针
4	pVarP	变量指针

10.17.2 处理逻辑

```
void PPLAT_SIP_TPMNG_MainEntry (IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara,
IN LPVOID pVarP)
{
    WORD16 wIndex = 0;
    switch( wState )
    {
    case PPLAT_SIP_TPMNG_INIT:      /* 初始化状态 */
        switch( wMsgId )
        {
```

```
        case EV_PPLAT_SIP_MASTER_POWER_ON:      /* 主板上电事件 */
            if (P_SIP_TPMNG_Init () == SIP_SUCCESS)
            {
                PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TPMNG_MASTER);
            }
            break;
        case EV_PPLAT_SIP_SLAVE_POWER_ON: /* 备板上电事件 */
            if (P_SIP_TPMNG_Init () == SIP_SUCCESS)
            {
                PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TPMNG_SLAVE);
            }
            break;
        default:
            PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TPMNG_INIT);
            break;
        }
        break;
    case PPLAT_SIP_TPMNG_MASTER   :     /* 工作状态 */
        /* 定时器超时事件处理 */
        if( wMsgId is timer)
        {
            dwResult=SIP_TPMNG_Proc_Timer(wState, wMsgId, pMsgPara, pVarp);
            Break;
        }

        switch( wMsgId )
        {
        case EV_PPLAT_SIP_TP_IP_MSG_BRSSOCKET_MSG: /* 来自底层的Socket消息 */
            P_SIP_TPMNG_onSocketRecvMsg((BYTE*)pMsgPara);
            break;
        case EV_PPLAT_SIP_MSG_IND: /* 来自SLB的上行消息选择处理进程直接透传 */
            P_SIP_TPMNG_onSlbRecvMsg((BYTE*)pMsgPara);
            break;
        case EV_PPLAT_SIP_MSG_REQ: /* 来自处理进程的下行SIP消息*/
            P_SIP_TPMNG_onPROCReq((BYTE*)pMsgPara);
            break;
        case EV_PPLAT_SIP_TP_DB_IPPORTCHG: /* IPPORT配置更改通知 */
            P_SIP_TP_IpPortChgNtf ((BYTE*)pMsgPara);
            break;
        case EV_PPLAT_SIP_TP_DB_LINKCHG: /* SIPLINK配置更改通知 */
            P_SIP_TP_LinkChgNtf((BYTE*)pMsgPara);
            break;
        case EV_PPLAT_SIP_MASTER_TO_SLAVE: /* 发给应用进程的倒换完成消息 */
            P_SIP_TP_onM2SProc ();
```

```
                PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TP_ST_SLAVE);
                break;
            case EV_PPLAT_SIP_AGT_DM_SIP_LINK_SHOW:
                PPLAT_SIP_onLinkquery((BYTE *)pMsgPara);
                break;
            /*IPSEC*/
            case EV_PPLAT_SIP_TPUI_CREATE_IPSEC_REQ:
                P_SIP_Create_Proc((PPLAT_SIP_TPUI_CREATE_IPSEC_REQ_T *)pMsgPara);
                break;
            case EV_PPLAT_SIP_TPUI_NOTIFY_IPSEC_REQ:
                P_SIP_Notify_Proc((PPLAT_SIP_TPUI_NOTIFY_IPSEC_REQ_T *)pMsgPara);
                break;
            case EV_PPLAT_SIP_TPUI_DELETE_IPSEC_REQ:
                P_SIP_Delete_Proc((PPLAT_SIP_TPUI_DELETE_IPSEC_REQ_T *)pMsgPara);
                break;
            /*IPSEC*/
            default:
                PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TP_ST_WORK);
                break;
            }
            PPLAT_SIP_OSS_SetDefaultNextState();
            break;
        case PPLAT_SIP_TPMNG_SLAVE:
            switch( wMsgId )
            {
            case EV_PPLAT_SIP_SLAVE_TO_MASTER: /*  发给应用进程的倒换完成消息  */
                P_SIP_TP_onS2MProc ();
                PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TP_ST_WORK);
                break;
            default:
                PPLAT_SIP_OSS_SetNextState(PPLAT_SIP_TP_ST_SLAVE);
                break;
            }
            PPLAT_SIP_OSS_SetDefaultNextState();
            break;
        default:
            break;
        }
    } } /* end of PPLAT_SIP_TP_Entry */
```

10.18 P_SIP_TPMNG_Init

10.18.1 概述

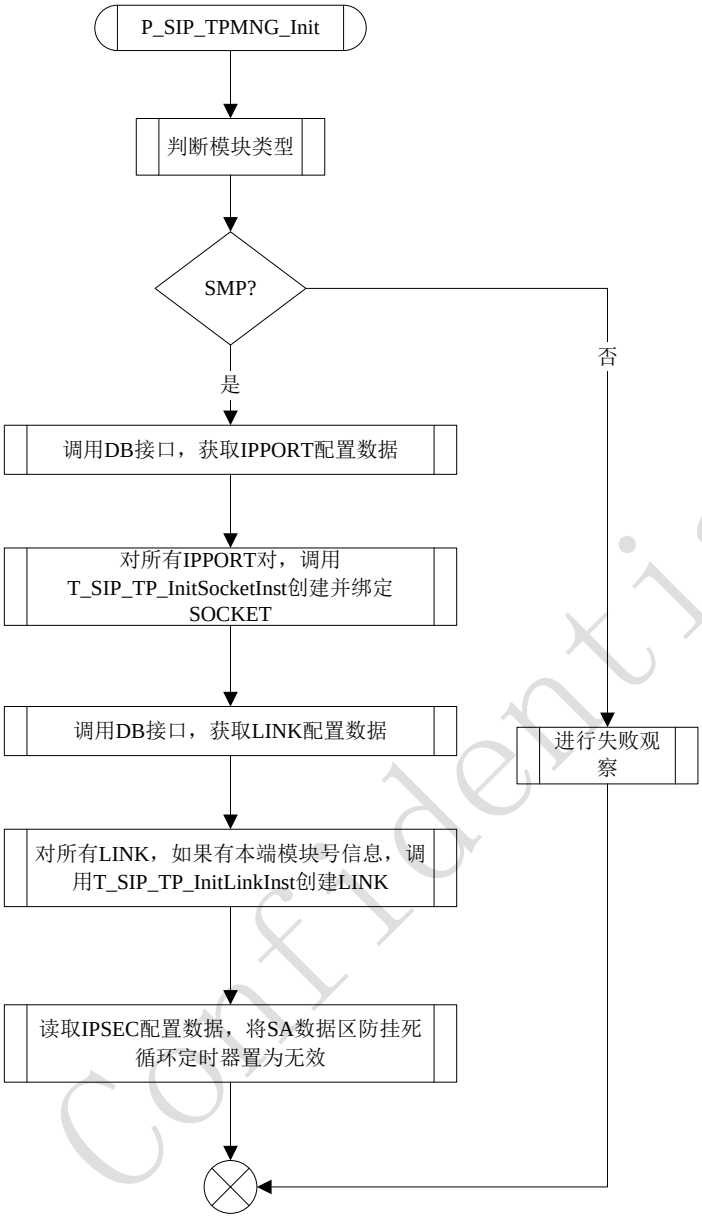
名称	P_SIP_TPMNG_Init	类型	TP 内部函数

功能		TP 收到上电消息后的处理。
性能		无
序号	分类	被调用函数
1	外	无
2	内	
序号	分类	使用的公共数据
1	外	无
2	内	
资源限制		
函数声明		
WORD32 P_SIP_TPMNG_Init(void)		

序号	分类	输入输出数据说明
3		

10.18.2 处理逻辑

本函数的处理逻辑如下图



10.19 SIP_TPMNG_Proc_Timer

10.19.1 概述

名称		SIP_TPMNG_Proc_Timer	类型	内部函数
功能		TP 收到定时器消息后的处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	无		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				

函数声明		
PPLAT_SIP_STATUS SIP_TPMNG_Proc_Timer (IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP)		
序号	分类	输入输出数据说明
1	I	wState：进程状态 PPLAT_SIP_TP_ST_INIT 0 PPLAT_SIP_TP_ST_WORK 1 PPLAT_SIP_TP_ST_SLAVE 2
2	I	wMsgId：事件号
3	I	pMsgPara：事件指针
4	I	pVarP：变量指针

10.19.2 处理逻辑

本函数的处理逻辑见下面的 PDL。

```
PPLAT_SIP_STATUS P_SIP_TP_TimerProc(IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP)
{
    BYTE                    bTpTimerEvent;
    BYTE                    bDataType;
    WORD32          dwTpExtParam = PPLAT_SIP_TIMER_EXTPRARAM_NULL;
    /*根据定时器消息号获取对应的 TP 数据区类型*/
    bDataType=PPLAT_SIP_TIMER_TP_getRelateDataType(wMsgId);

    /*如果是 SOCKET 数据区*/
    if( bDataType == PPLAT_SIP_DATAAREA_SOCKET)
    {
        /*获取 SOCKET 定时器扩展参数和内部功能定时器超时事件*/
        if(!PPLAT_SIP_TIMER_TP_convertSocketInst(wMsgId, pMsgPara,&bTpTimerEvent,&dwTpExtParam))
        {
            return PPLAT_SIP_ERROR_TP_TIMER;
        }
    }

    /*如果是 LINK 数据区*/
    else if( bDataType == PPLAT_SIP_DATAAREA_LINK)
    {
        /*获取 LINK 定时器扩展参数和内部功能定时器超时事件*/
        if(!PPLAT_SIP_TIMER_TP_convertLinkInst(wMsgId, pMsgPara,&bTpTimerEvent,&dwTpExtParam))
        {
            return PPLAT_SIP_ERROR_TP_TIMER;
        }
    }
```

```
        /*如果是 SA 数据区*/
        else if( bDataType == PPLAT_SIP_DATAAREA_IPSEC)
        {
            /*获取 LINK 定时器扩展参数和内部功能定时器超时事件*/
            if(!PPLAT_SIP_TIMER_TP_convertIPSecSAInst(wMsgId,
pMsgPara,&bTpTimerEvent,&dwTpExtParam))
            {
                return PPLAT_SIP_ERROR_TP_TIMER;
            }
        }

        /*如果与数据区无关*/
        else if( bDataType == PPLAT_SIP_TIMERTYPE_NORELATEDATA)
        {
            /*获取内部功能定时器超时事件*/
            if(!PPLAT_SIP_TIMER_TP_convertNoParam(wMsgId, pMsgPara,&bTpTimerEvent))
            {
                return PPLAT_SIP_ERROR_TP_TIMER;
            }
        }

        /*如果是 CID 数据区*/
        else if( bDataType == PPLAT_SIP_DATAAREA_CID)
        {
            /*获取 CID 定时器扩展参数和内部功能定时器超时事件*/
            if(!PPLAT_SIP_TIMER_TP_convertCIDInst(wMsgId,
pMsgPara,&bTpTimerEvent,&dwTpExtParam))
            {
                return PPLAT_SIP_ERROR_TP_TIMER;
            }
        }

        /*如果是 DID 数据区*/
        else if( bDataType == PPLAT_SIP_DATAAREA_DID)
        {
            /*获取 DID 定时器扩展参数和内部功能定时器超时事件*/
            if(!PPLAT_SIP_TIMER_TP_convertDIDInst(wMsgId,
pMsgPara,&bTpTimerEvent,&dwTpExtParam))
            {
                return PPLAT_SIP_ERROR_TP_TIMER;
            }
        }

        /*其余情况,数据区类型不正确,返回失败*/
```

```
else
{
    return PPLAT_SIP_ERROR_TP_TIMER;
}

/*对应不同的内部功能定时器超时事件进行分别处理*/
switch(bTpTimerEvent)
{
case EV_PPLAT_SIP_TIMEROUT_TP_SOKCETINIT:
    if(P_SIP_TP_InitSocketInst(dwTpExtParam)!=PPLAT_SIP_SUCCESS)
    {
        return PPLAT_SIP_ERROR_TP_TIMER;
    }
 break;

case EV_PPLAT_SIP_TIMEROUT_TP_LINKINIT:
    if(P_SIP_TP_InitLinkInst(dwTpExtParam)!=PPLAT_SIP_SUCCESS)
    {
        return PPLAT_SIP_ERROR_TP_TIMER;
    }
    break;

case EV_PPLAT_SIP_TIMEROUT_TP_LINKIDLEPROTECT:
    P_SIP_TP_LinkIdleTimerEvent(dwTpExtParam);
    break;

case EV_PPLAT_SIP_TIMEROUT_TP_LINKHEARTBEAT:
    P_SIP_TP_HtCheckTimerEvent(dwTpExtParam);
    break;

case EV_PPLAT_SIP_TIMEROUT_TP_LINKHEARTBEATRSP:
    P_SIP_TP_RespTimerEvent(dwTpExtParam);
    break;

case EV_PPLAT_SIP_TIMEROUT_TP_IPSECSAWAITREL:
    P_SIP_TP_SaDelDelayTimerEvent(dwTpExtParam);
    break;

case EV_PPLAT_SIP_TIMEROUT_TP_SOCKETPROTECT:
    if(P_SIP_TP_TcpConnTimerEvent(dwTpExtParam)!=PPLAT_SIP_SUCCESS)
    {
    }
    break;
```

```
        case EV_PPLAT_SIP_TIMEROUT_TP_SAPROTECT:
            if(P_SIP_TP_IPSecTimerProt()!=PPLAT_SIP_SUCCESS)
            {
            }
            break;

        case EV_PPLAT_SIP_TIMEROUT_TP_SIGCOMPCID:
if(SIP_SIGCOMP_Timer_Proc(bTpTimerEvent,dwTpExtParam)!=PPLAT_SIP_SUCCESS)
            {
            }
            break;

        case EV_PPLAT_SIP_TIMEROUT_TP_SIGCOMPDID:
if(SIP_SIGCOMP_Timer_Proc(bTpTimerEvent,dwTpExtParam)!=PPLAT_SIP_SUCCESS)
            {
            }
            break;

        default:
            break;
        }
        return PPLAT_SIP_SUCCESS;
}
```

10.20 P_SIP_TPMNG_onSocketRecvMsg

10.20.1 概述

名称		P_SIP_TPMNG_onSocketRecvMsg	类型	内部函数
功能		TP 收到 IP 承载的上行消息后的处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	无		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
PPLAT_SIP_STATUS P_SIP_TPMNG_onSocketRecvMsg e(BYTE *pMsg)				
序号	分类	输入输出数据说明		
1	I	pMsg：输入消息指针		

10.20.2 处理逻辑

本函数的处理逻辑见下面的 PDL。

```
PPLAT_SIP_STATUS P_SIP_TPMNG_onSocketRecvMsg (BYTE *pMsg)
{
    memcpy(&tSocketMsg, pMsg, sizeof(BRS_SOCKET_IO_MSG));

    switch (tSocketMsg.type)      /* 判断 Socket 消息类型*/
    {
    case BRS_SO_CONNECTED:
        T_SIP_TP_TcpConnProc(pMsg);
        break;
    case BRS_SO_ACCEPTED:
        T_SIP_TP_TcpAccptProc(pMsg);
        break;
    case BRS_SO_DATA:
        T_SIP_TPMNG_SocketDataRecvInd(pMsg);
        break;
    case BRS_SO_CLOSE:
        T_SIP_TP_TcpCloseProc(pMsg);
        break;
    case BRS_SO_ERROR:
        break;
    default:
        break;
    }
}
```

10.21 P_SIP_TPMGN_SocketDataRecvInd

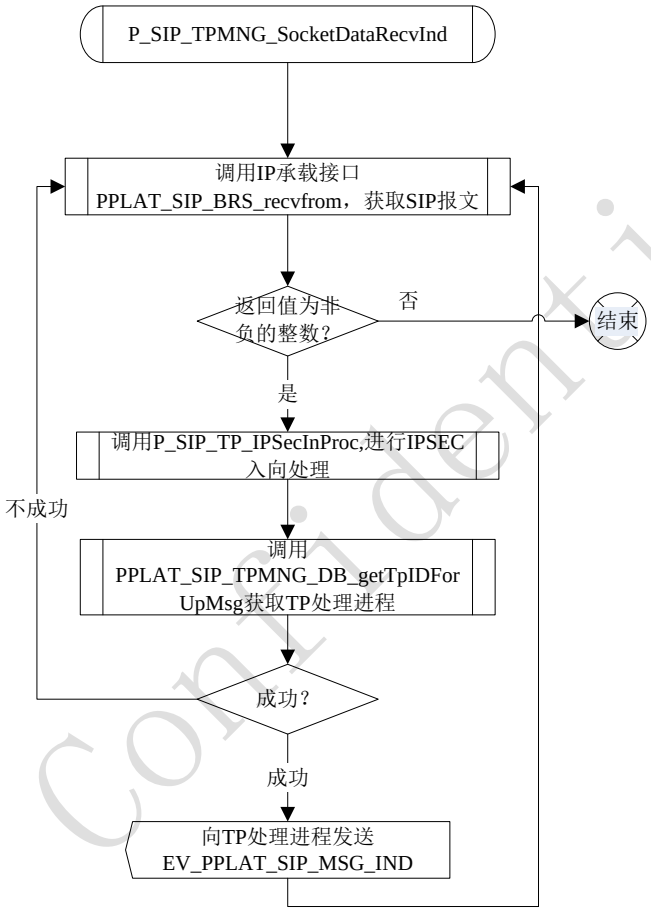
10.21.1 概述

名称		P_SIP_TPMNG_SocketDataRecvInd	类型	内部函数
功能		TP 管理进程收到上行报文后，进行上行处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	无		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
PPLAT_SIP_STATUS P_SIP_TPMNG_SocketDataRecvInd (BYTE *pMsg)				
序号	分类	输入输出数据说明		

1	I	pMsg：输入消息指针

10.21.2 处理逻辑

本函数的处理逻辑如下图



10.22 P_SIP_TPMNG_onSlbRecvMsg

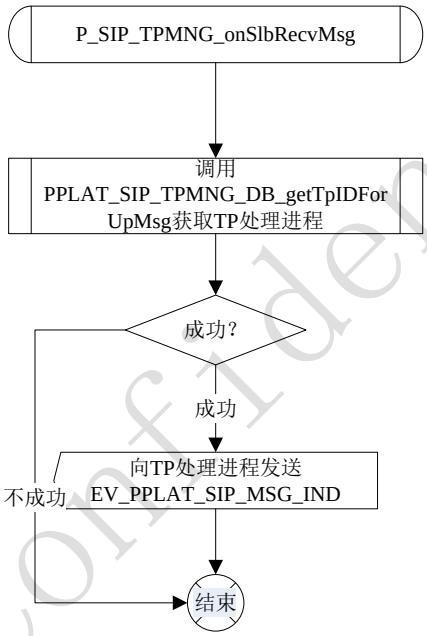
10.22.1 概述

名称		P_SIP_TPMNG_onSlbRecvMsg	类型	内部函数
功能		TP 收到 SLB 的上行消息后，选择处理进程进行透传处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	无		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				

函数声明		
PPLAT_SIP_STATUS P_SIP_TPMNG_onSlbRecvMsg (BYTE *pMsg)		
序号	分类	输入输出数据说明
1	I	pMsg：输入消息指针

10.22.2 处理逻辑

本函数的处理逻辑如下图



10.23 P_SIP_TPMNG_onPROCReq

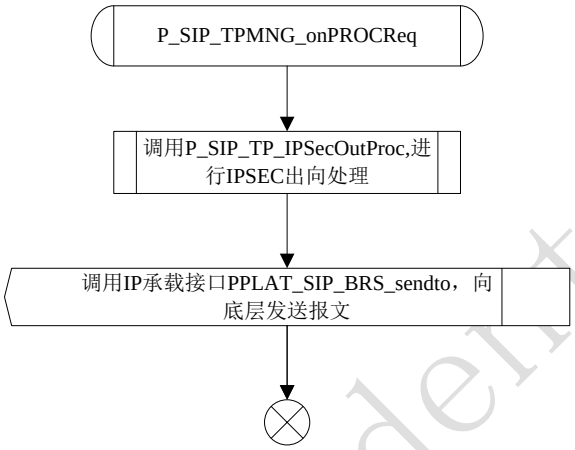
10.23.1 概述

名称		P_SIP_TPMNG_onPROCReq	类型	内部函数
功能		TP 收到处理进程的下行消息后，进行下行处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	无		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
PPLAT_SIP_STATUS P_SIP_TPMNG_onPROCReq (BYTE *pMsg)				

序号	分类	输入输出数据说明
1	I	pMsg：输入消息指针

10.23.2 处理逻辑

本函数的处理逻辑如下图



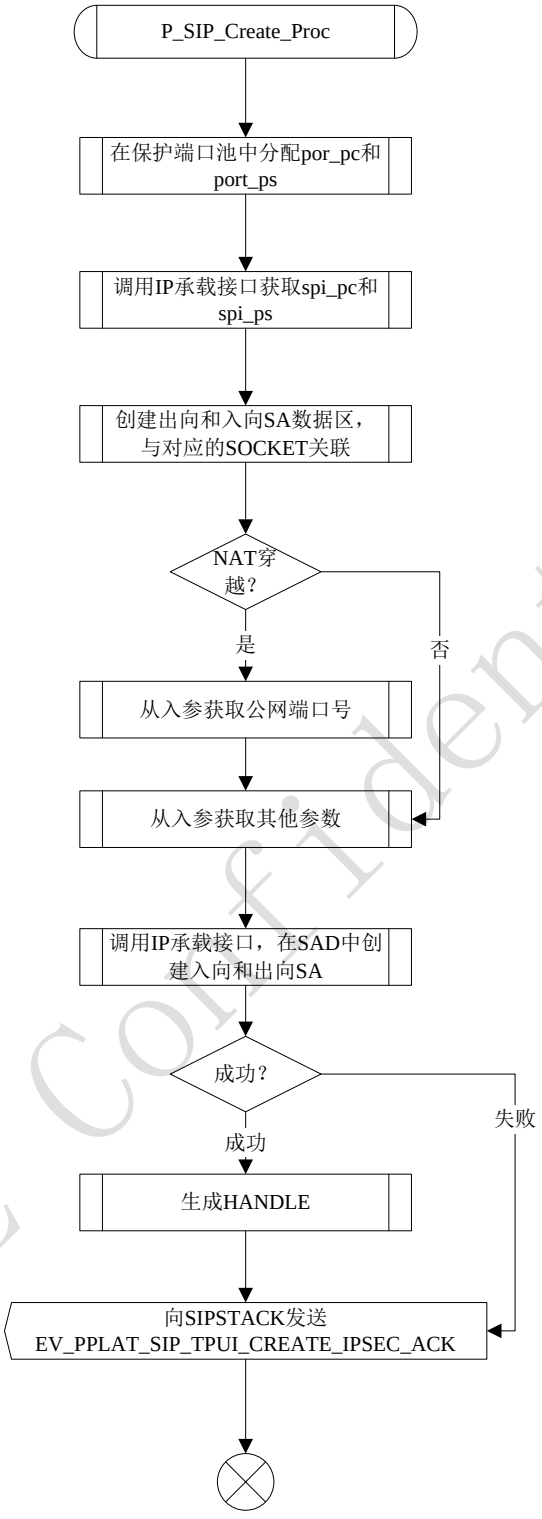
10.24 P_SIP_Create_Proc

10.24.1 概述

名称		P_SIP_Create_Proc	类型	内部函数
功能		传输适配层收到 SIPSTACK 的创建 IPSEC SA 请求后的处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	无		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
PPLAT_SIP_STATUS P_SIP_Create_Proc (PPLAT_SIP_TPUI_CREATE_IPSEC_REQ_T *ptSipCrtIPSecReq)				
序号	分类	输入输出数据说明		
1	I	ptSipCrtIPSecReq：创建 IPSEC 请求结构指针，结构见第 7 章		

10.24.2 处理逻辑

本函数的处理逻辑如下图



10.25 P_SIP_Notify_Proc

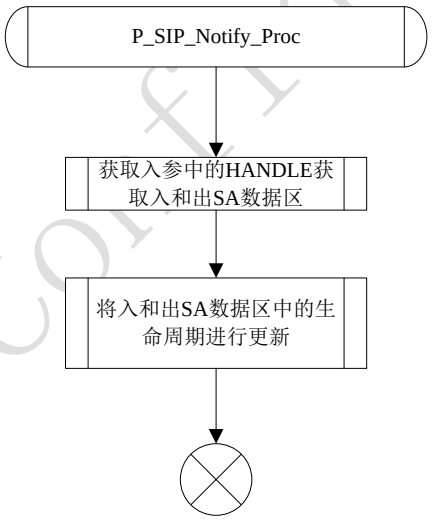
10.25.1 概述

名称	P_SIP_Notify_Proc	类型	内部函数
功能	传输适配层收到 SIPSTACK 的 SA 通知后的处理		

性能		无
序号	分类	被调用函数
1	外	无
2	内	无
序号	分类	使用的公共数据
1	外	无
2	内	无
资源限制		
函数声明		
PPLAT_SIP_STATUS P_SIP_Notify_Proc (PPLAT_SIP_TPUI_NOTIFY_IPSEC_REQ_T *ptSipNotiIPSecReq)		
序号	分类	输入输出数据说明
1	I	ptSipUdtIPSecReq：SA 通知请求结构指针，结构见第 7 章

10.25.2 处理逻辑

本函数的处理逻辑如下图



10.26 P_SIP_Delete_Proc

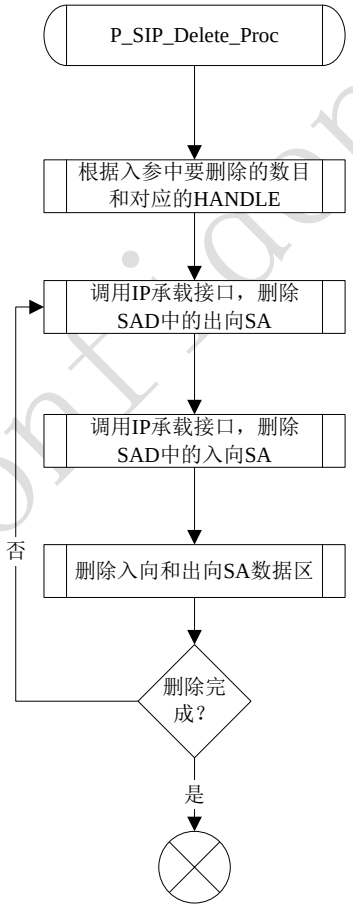
10.26.1 概述

名称	P_SIP_Delete_Proc	类型	内部函数
功能	传输适配层收到 SIPSTACK 的删除 IPSEC SA 请求后的处理		
性能	无		
序号	分类	被调用函数	
1	外	无	
2	内	无	
序号	分类	使用的公共数据	
1	外	无	

2	内	无
资源限制		
函数声明		
PPLAT_SIP_STATUS P_SIP_Delete_Proc (PPLAT_SIP_TPUI_DELETE_IPSEC_REQ_T *ptSipDelIPSecReq)		
序号	分类	输入输出数据说明
1	I	ptSipDelIPSecReq：删除 IPSEC SA 请求结构指针，结构见第 7 章

10.26.2 处理逻辑

本函数的处理逻辑如下图



10.27 P_SIP_TP_IpPortChgNtf

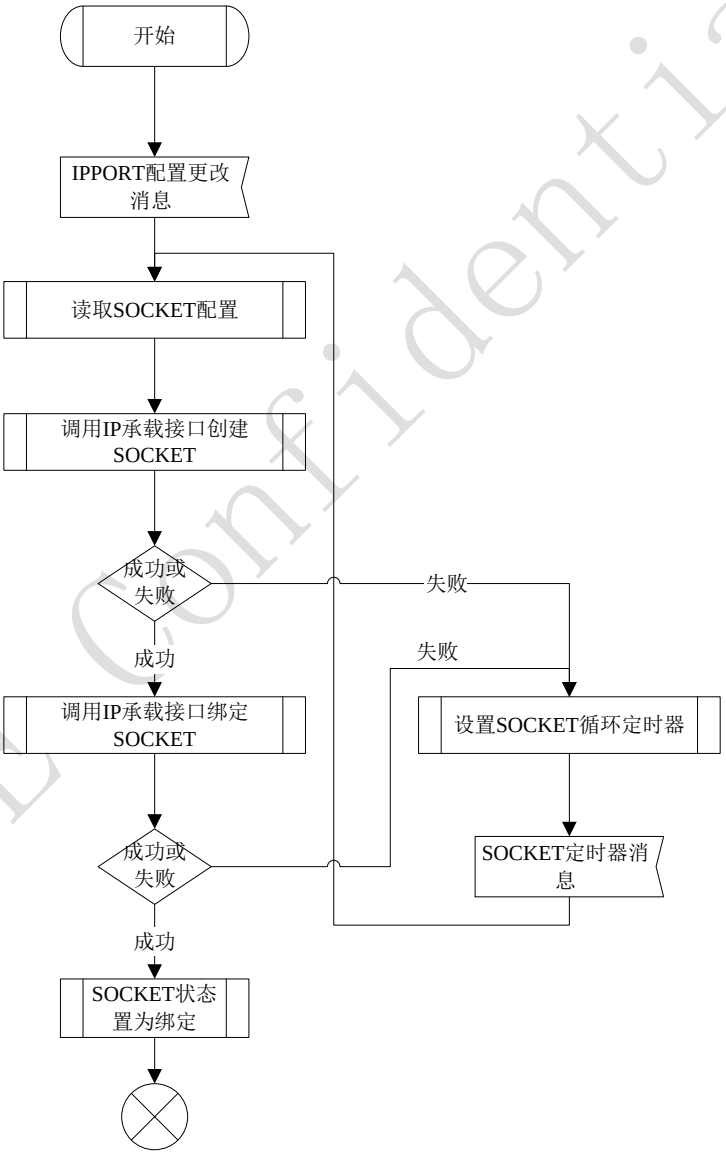
10.27.1 概述

名称	P_SIP_TP_IpPortChgNtf	类型	IPPORT 配置更改处理函数
功能	对 IPPORT 配置更改消息的处理。		
性能	无		
序号	分类	被调用函数	
1	外	无	

2	内	无
序号	分类	使用的公共数据
1	外	无
2	内	无
资源限制		
无		
函数声明		
PPLAT_SIP_STATUS P_SIP_TP_IpPortChgNtf(BYTE* pMsg)		
序号	分类	输入输出数据说明
1	I	pMsg：输入消息指针

10.27.2 处理逻辑

本函数的处理逻辑如下图



10.28 P_SIP_TP_LinkChgNtf

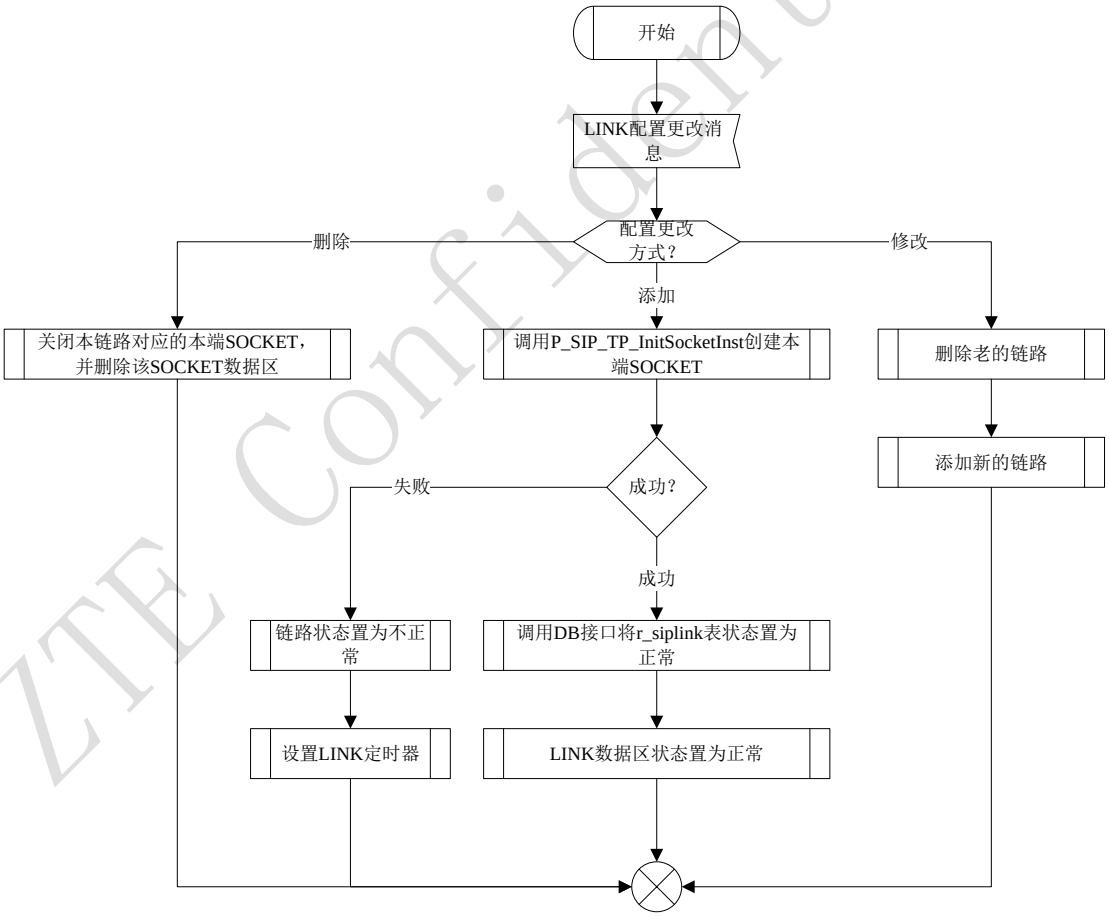
10.28.1 概述

名称	P_SIP_TP_LinkChgNtf	类型	链路配置更改处理函数

功能		对链路配置更改消息的处理。
性能		无
序号	分类	被调用函数
1	外	无
2	内	无
序号	分类	使用的公共数据
1	外	无
2	内	无
资源限制		
无		
函数声明		
PPLAT_SIP_STATUS P_SIP_TP_LinkChgNtf (BYTE* pMsg)		
序号	分类	输入输出数据说明
1	I	pMsg：输入消息指针

10.28.2 处理逻辑

本函数的处理逻辑如下图



10.29 SIP_TU_Inv2XXHandle

/* 修改函数实现 */

10.29.1 概述

名称		SIP_TU_ Inv2XXHandle	类型	TU 内部函数
功能		TU 收到 INVITE 的 2xx 响应后的处理。		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内			
资源限制				
函数声明				
WORD32 SIP_TU_Inv2XXHandle (SIP_TU_LEG_T *ptLeg, T_SIP_MSG *ptMsg)				

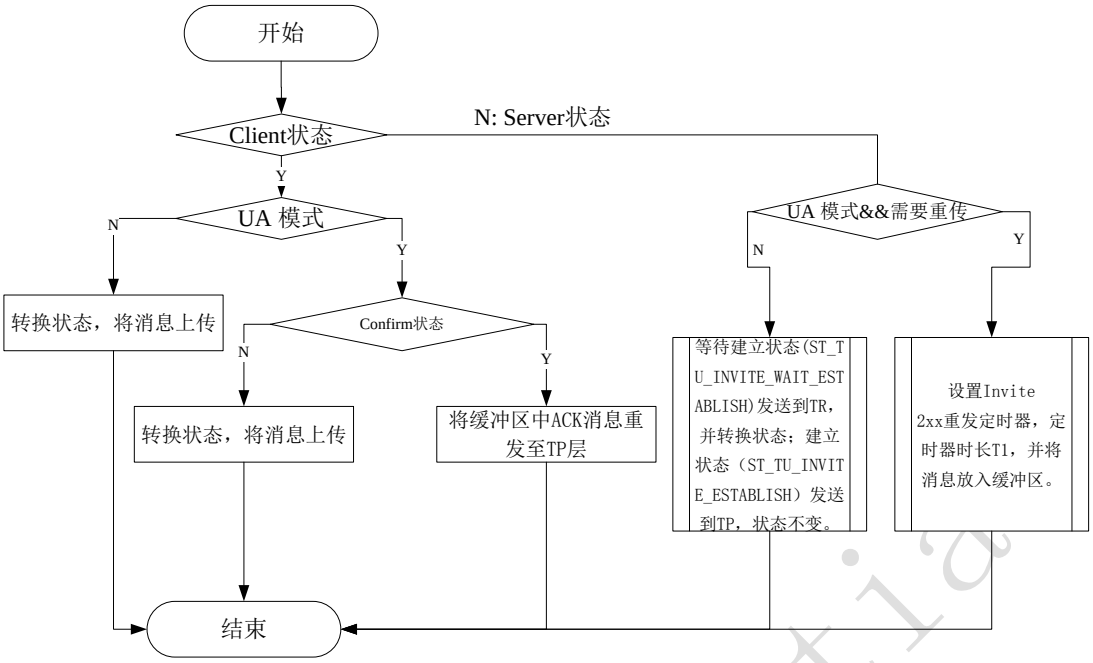
序号	分类	输入输出数据说明
1	ptLeg	TU 数据区
2	ptMsg	消息
3		

10.29.2 处理逻辑

- 传入 TU 数据区实例
- 上行（UAC）：Proxy 模式：上传响应，由业务转发

  UA 模式：读 2xx 重传吸收配置，如果配置为吸收，则停止上传消息，将缓冲区中的 ACK 消息重新发送至 TP 层。如果配置为不吸收，则将 2xx 响应上报到业务。如果 ACK 标记为不重发，则丢弃消息或报错。
- 下行（UAS）：Proxy 模式：等待建立状态(ST_TU_INVITE_WAIT_ESTABLISH)发送到 TR，并转换状态；建立状态（ST_TU_INVITE_ESTABLISH）发送到 TP，状态不变。

  UA 模式：数据区在等待建立状态(ST_TU_INVITE_WAIT_ESTABLISH)或建立状态(ST_TU_INVITE_ESTABLISH)，判断消息中的重传标志，如果是需要重传，设置 Invite 2xx 重发定时器，定时器时长 T1，并将消息放入缓冲区。



10.30 SIP_TU_CALL_AckProcOfEstablish

10.30.1 概述

名称		SIP_TU_CALL_ AckProcOfEstablish	类型	内部函数
功能		对呼叫的 ST_TU_INVITE_ESTABLISH 态下收到 INVITE 的 ACK 的处理		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	SIP_TU_SendMsg		
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
WORD32 SIP_TU_CALL_ AckProcOfEstablish (IN void* ptLeg, IN SIP_TU_EVENT_E eSipEvent, IN SIP_TU_MSG_T* ptMsg)				
序号	分类	输入输出数据说明		
1	ptLeg	Leg 指针		
	eSipEvent	TU 内部事件号		
	ptMsg	TU 的内部 MSG 结构指针		

10.30.2 处理逻辑

1． 当作为UA收到事务层发来的ACK请求，UAS就需要杀掉INVITE的2XX响应的重发定时器，并释放消息

2． 当作为UA收到上层应用发来的ACK，读取是否吸收2xx响应配置，如果吸收则缓存ACK消息，同时在数据内保存ACK消息的消息ID

3． 若为上行的消息，设置消息的事件属性为EV_PPLAT_SIP_API_SUBSEQUENT_IND

4. 调用SIP_TU_SendMsg发送ACK消息

10.31 SIP_TU_onResend1xxTimer

10.31.1 概述

名称		SIP_TU_onResend1xxTimer	类型	内部函数
功能		TU 收到支撑的重发 1xx 定时器超时事件的处理函数		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	SIP_TU_CALL_ProcessMachine		
序号	分类	使用的公共数据		
1	外	无		
2	内			
资源限制				
函数声明				
WORD32 SIP_TU_onResend1xxTimer (IN void* pvIn)				
序号	分类	输入输出数据说明		
1	pvIn	入参 结构指针		

10.31.2 处理逻辑

本函数的处理逻辑见下 ：

1. 分发到 SIP_TU_xxx_ProcessMachine 中，其中内部事件为 SIP_TU_EVENT_RESEND_1xx_TIMEOUT，且 ptMsg 为 NULL

10.32 PPLAT_SIP_MainEntry

10.32.1 概述

名称		PPLAT_SIP_MainEntry	类型	主函数
功能		TU TR 模块进程入口函数		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内	P_SIP_Stack_Init P_SIP_Tulayer_dispatchEvent P_SIP_TrLayer_dispatchEvent P_SIP_Proc_Timer, P_SIP_CommAgent_dispatchEvent P_SIP_MasterSlave_dispatchEvent		
序号	分类	使用的公共数据		
1	外	无		
2	内	SIP_SAP_ID		
资源限制				
函数声明				

void PPLAT_SIP_MainEntry (IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP);		
序号	分类	输入输出数据说明
1	wState	进程状态 ST_STACK_INIT 0 ST_STACK_MASTER 1 ST_STACK_SLAVE 2
2	wMsgId	事件号
3	pMsgPara	事件指针
4	pVarP	变量指针

10.32.2 处理逻辑

在原有的处理逻辑中，在主板工作态下增加对收到信令跟踪任务设置事件、信令跟踪任务取消事件，失败观察任务设置事件、失败观察任务取消事件的处理。

10.33 PPLAT_SIP_TP_Entry

10.33.1 概述

名称		PPLAT_SIP_TP_Entry	类型	主函数
功能		SIP TP 模块进程入口函数		
性能		无		
序号	分类	被调用函数		
1	外	无		
2	内			
序号	分类	使用的公共数据		
1	外	无		
2	内	无		
资源限制				
函数声明				
void PPLAT_SIP_TP_Entry(IN WORD16 wState, IN WORD16 wMsgId, IN LPVOID pMsgPara, IN LPVOID pVarP);				
序号	分类	输入输出数据说明		
1	I	wState：进程状态 PPLAT_SIP_TP_ST_INIT 0 PPLAT_SIP_TP_ST_WORK 1 PPLAT_SIP_TP_ST_SLAVE 2		
2	I	wMsgId：事件号		
3	I	pMsgPara：事件指针		
4	I	pVarP：变量指针		

10.33.2 处理逻辑

在 TP 主板工作态下增加对于对收到信令跟踪任务设置事件、信令跟踪任务取消事件，失败观察任务设置事件、失败观察任务取消事件的处理。

增加对于收到 EV_PPLAT_SIP_MSG_IND 上行事件的处理。

11 兼容性说明

注：修订记录编号 0002

本节描述，SIP 平台核心代码的修改，对于兼容性的考虑。

11.1 添加 record-route 与修改 record-route

对于 CSCF V4.03 项目，需要指定后续请求所要发送的端口（即本端的接入端口），通过 PPLAT_SIP_DB_GetSubAccessAddr 接口获取后续所要发送的端口，根据该信息进行添加、更改 record-route 等操作。

是否添加、更改 record-route 中的端口信息，可以通过在 PPLATAPP 中增加项目开关项进行控制，不需要添加更改时，将开关关闭即可。因此对 SSS2.0 项目及原有 CSCF 40 版本兼容性没有影响。

此开关项定义如下，是否打开有项目进行选择。

```
/*  是否在 RECORD-ROUTE 中添加 PORT 参数  */
#define PPLAT_SIP_ADD_RECORDROUTE_PORT                  TRUE

/*  是否修改 RECORD-ROUTE 中 IP-PORT 参数  */
 #define PPLAT_SIP_MODIFY_RECORDROUTE_PORT              TRUE
```

12 代码规划

本模块规划的所有源代码文件见表 12.1。

表11. 1

序号	源代码文件名称	文件内容
1	pplat_sip_tu_mng.h	SIP 平台 tu 管理进程声明头文件
2	pplat_sip_tu_Mng.c	SIP 平台 tu 管理进程实现文件
3	pplat_sip_tp_mng.h	SIP 平台 tp 管理进程声明头文件
4	pplat_sip_tp_mng.c	SIP 平台 tp 管理进程实现文件。
5	pplat_sip_omm.c	SIP 平台操作维护接口缺省实现文件
6	pplat_sip_dba.c	SIP 平台数据库接口缺省实现文件
7	pplat_sip_allocator.h	SIP 平台内存分配器接口声明文件
8	pplat_sip_allocator.c	SIP 平台内存分配器实现文件

13 参考资料

[1] RFC3261：SIP: Session Initiation Protocol，June 2002。