TSN 分布式控制器设计文档

（版本 1.1）

OpenTSN 开源项目组

2021 年 5 月

版本历史

版本	修订时间	修订内容	修订人	文件标识
1.0	**2020.8**	初版编制	李军帅	**TSNLight2.0 分布式控制器**
1.1	**2021.5**	对 1.0 版本进行修改，主要修改格式	李军帅	

目录

1、设计目标

为了使枫林 TSN 能够正常运行，开发 TSN 分布式控制器，与枫林 TSN 配套使用。该控制器为分布式控制器 TSNLight2.0，结合 OpenTSN2.0 使用。

2、需求分析

设计需求包含以下几个方面：

- 需要对枫林 TSN 进行配置
- 需要处理 PTP 时间同步报文
- 需要把报文转换成枫林 TSN 能够识别的报文
- 需要处理 ARP 请求报文

根据设计需求，TSN 控制器需要完成以下功能：

- 数据接收和数据发送：能够与枫林 TSN 进行报文的接收与发送
- 本地管理：能够构造特定的配置报文，对枫林 TSN 进行配置
- PTP 时间同步：确定主从时钟，主时钟能够定时构造 sync 报文和处理 req 报文，从时钟能够处理 sync 报文和 resq 报文，并且能够根据时间戳计算出主从时间偏差 offset，并发送配置报文对从时钟进行校正
- 流映射管理：对未经映射的报文查表映射，替换报文的目的 mac 为 TSNTag，使枫林 TSN 能够根据 TSNTag 识别报文类型，并进行查表转发

- 重映射：把报文的 TSNTag 替换成原始的目的 mac 地址，使到达接收端的报文与发送端发送的报文一致
- ARP 代理：处理 ARP 请求报文，并直接构造 ARP 响应报文进行响应

3、总体设计

3.1 系统结构

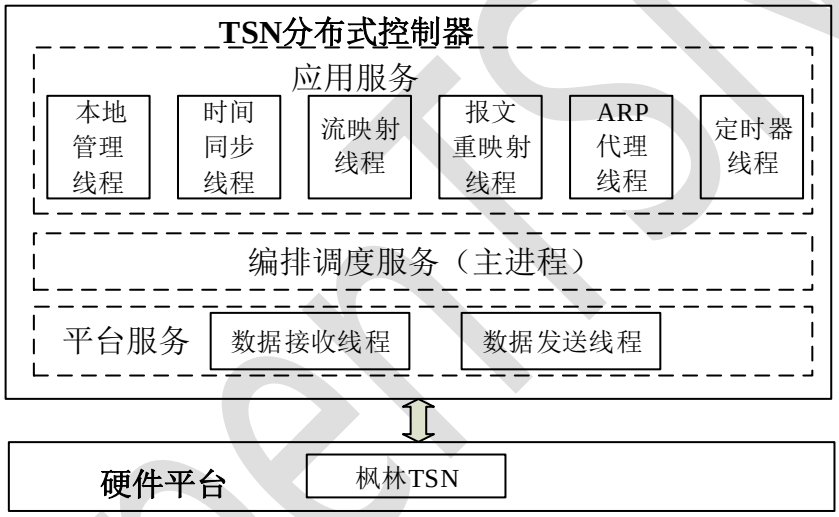


图 3-1 总体架构框图

TSN 分布式控制器为单进程多线程工作模式，软件系统结构如上图所示，主要功能包含三部分，由下自上依次为：平台服务、编排调度服务和应用服务。

平台服务提供平台基础服务，为 TSN 分布式控制器与与硬件平台的通信提供统一的入口和出口。与硬件平台相关，可为上层的应用服务屏蔽硬件底层的差异。主要包括数据接收线程和数据发送线程，负责从硬件接收数据报文和将数据报文发送至硬件。

编排调度服务负责各个线程间通信消息的编排和对各个服务线程的管理调度，为平台服务线程与应用服务线程、平台服务线程与平台服务线程、应用服务线程与应用服务线程间通信的桥梁。

应用服务提供用户定制服务，与用户具体应用相关，每一个应用服务为一个应用线程，具有良好的可扩展性。主要包括本地管理线程、时间同步线程、流映射线程、报文重映射线程、arp 代理线程和定时器线程。

本地管理线程实现对枫林 TSN 的配置；时间同步线程协同枫林 TSN 实现 1588 时钟同步协议；流映射线程实现对未经映射的报文查表映射成枫林 TSN 能够识别的带 TSNTag 标记位的报文；报文重映射线程负责把带 TSNTag 标记位的报文还原成原始数据报文； ARP 代理线程处理 ARP 请求报文；定时器线程负责整个 TSN 分布式控制器的周期定时应用。

3.2 实现架构

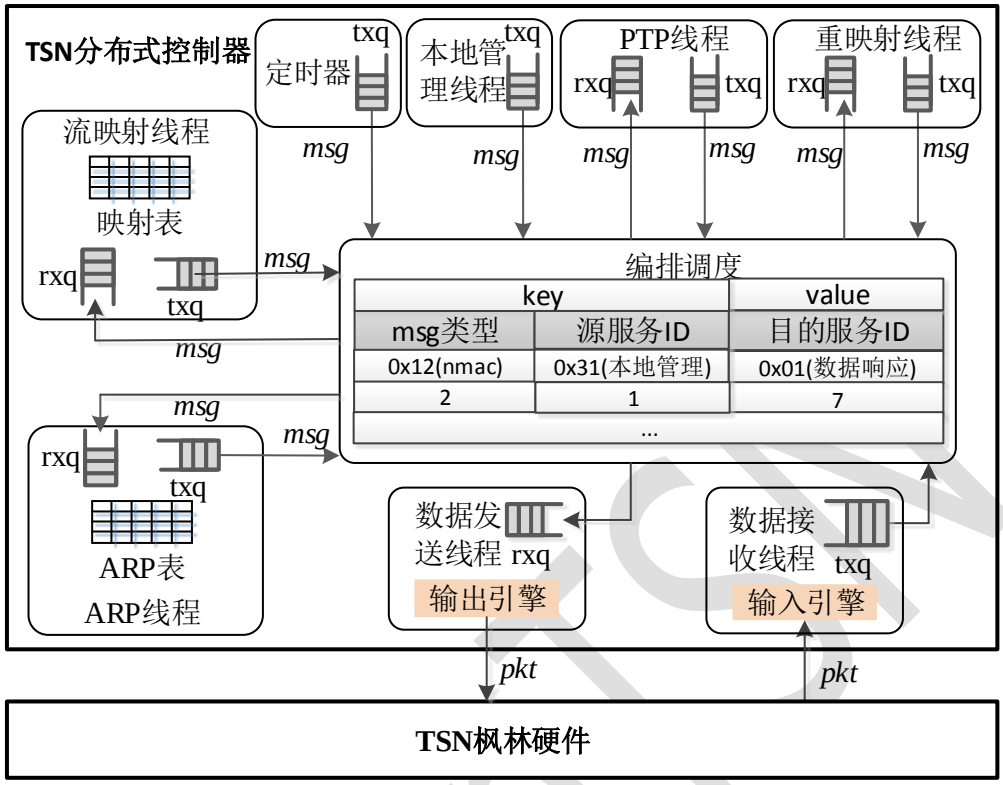


图 3-2 实现架构框图

TSN 分布式控制器实现架构如上图所示。TSN 枫林硬件平台实现。TSN 分布式控制器主要实现以下几个基本功能，分别为数据接收和发送、本地配置管理、PTP 时间同步、流映 •射管理、报文重映射、ARP 代理和定时器。与 TSN 枫林硬件平台协同处理，实现对 TSN 枫林进行参数配置、主从节点时间同步、ST 流的规划调度、非 ST 流的管理转发以及报文重映射和 ARP 代理等功能。

TSN 分布式控制器为单进程多线程工作模式。TSN 控制器进程内部的线程间通信使用自定义的 msg 消息，msg 类似于 skb_buf 结构，包含控制信息和报文地址两部分。msg 数据结构的详细描述参看附录。

rxq 为接收队列，txq 为发送队列。所有的服务线程都有一个线程

编号，用于唯一标识线程的身份。

编排调度服务负责各个线程间通信消息的编排，由主进程实现。通过依次轮询各个服务线程的 txq 队列并取出 msg 消息， 然后根据编排策略将 msg 消息送入相应的服务线程的 rxq 队列。

编排策略为事先离线规划好，TSN 分布式控制器在系统初始化后本地存储。从 txq 队列取出 msg 消息后，根据 msg 的控制信息字段 msg 类型和源服务 ID 拼接成 key，然后查找编排策略表获取 value 值，即目的服务 ID，再把 msg 消息送入目的服务 ID 所对应的服务线程的 rxq 队列。

数据接收线程的主要功能是接收硬件平台送往 CPU 的报文并处理，数据发送线程的主要功能是将软件报文发往至硬件平台。由于硬件平台的差异性会导致从硬件收报文至软件和从软件发送报文至硬件的实现方式有所差异，另外应用场景的差异也会导致对于收到的数据报文和要发往硬件的数据报文处理方式有所差异。因此在数据接收线程有一个输入引擎，数据发送线程有一个输出引擎，以适配不同的硬件平台和应用需求。

数据接收线程，通过输入引擎从硬件接收报文和构造 msg 消息，并根据数据报文 metedata 对 msg 相关域进行赋值，然后写入自己的 txq 队列。

数据接收线程通过轮询自己的 rxq 队列，依次取出 msg 消息，然后将 msg 交由输出引擎模块，最后将 msg 所承载的报文发送至硬件平台。

应用服务线程感知不到其他服务线程的存在。以流映射线程为例，流映射线程只需要实现流映射功能即可。通过轮询自己的 rxq 队列，依次取出 msg 消息并根据 msg 的报文地址获取数据报文，然后对数据报文进行查表映射成枫林 TSN 能够识别的带 TSNTag 标记位的报文。最后构造 msg 消息，并将报文处理结果写入 msg 控制信息相应字段，msg 消息写入自己的 txq 队列。

3.3 工作流程

3.3.1 整体流程图

整体流程图如下所示。

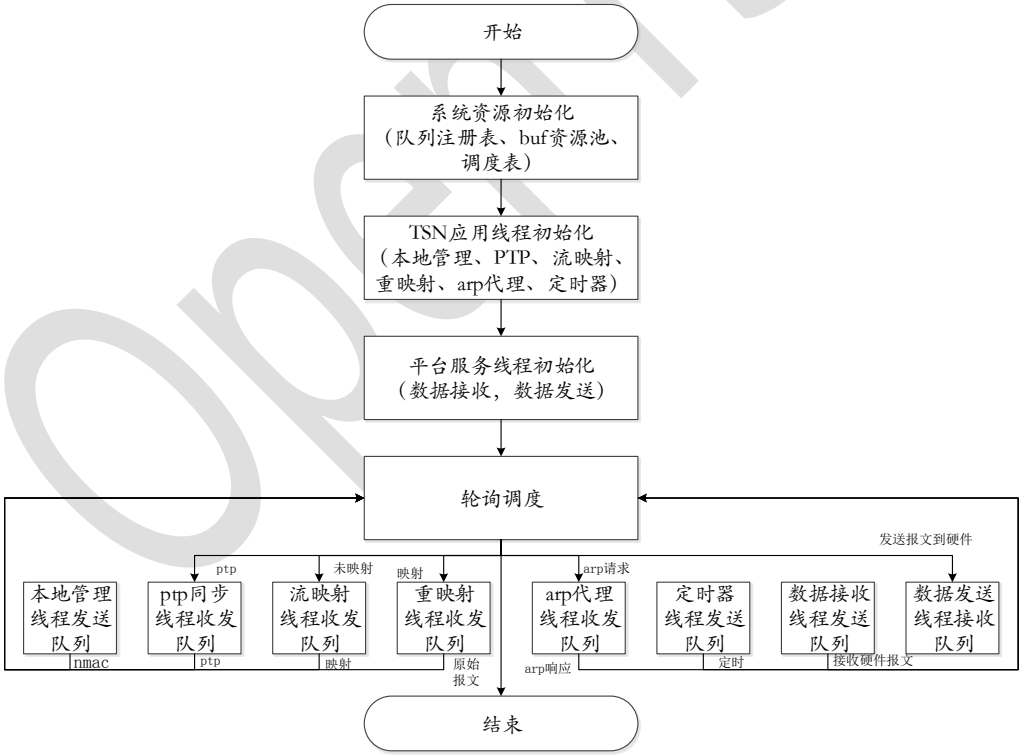


图 3-3 整体流程图

TSN 分布式控制器采用单进程多线程架构

主线程

（1）系统资源初始化：包括收发队列注册表的初始化，buf 地址资源池的初始化以及调度表的初始化。

（2）TSN 应用线程初始化：依次初始化本地管理线程、PTP 时间同步线程、流映射线程、重映射线程、ARP 代理线程和定时器线程，完成各个应用线程所依赖的系统资源的初始化。

（3）平台服务线程初始化：依次初始化数据接收线程和数据发送线程，完成数据接收、发送线程所依赖的系统资源的初始化。

（4）轮询调度：依次轮询各个服务线程的 txq 队列并取出 msg 消息，然后根据编排策略将消息送入相应的 rxq 队列。

3.3.2 本地配置处理流程

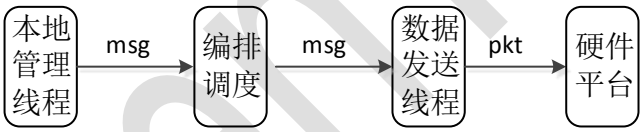


图 3-4 本地配置处理流程图

（1）本地管理线程：本地管理线程根据配置信息生成配置报文，发送到本地管理的发送队列中。

（2）编排调度：编排调度读取本地管理线程的发送队列，取出 msg，根据 msg 信息（msg 类型和源服务 ID）查找编排调度表，查找的结果为目的服务 ID 是数据发送线程，然后将 msg 发送到数据发送线程的 rxq 队列。

（3）数据发送线程：msg 到达数据响应线程后，依次取出 rxq 队列中的 msg，并将 msg 所承载的报文发送到硬件平台。

3.3.3 时间同步处理流程

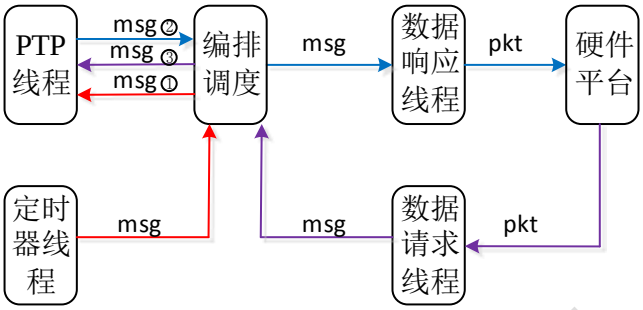


图 3-5 时间同步处理流程图

时间同步流程分为主和从，主设备需要有定时期线程的参与，用于定时发送 sync 报文，从设备不需要定时器线程参与。

主时钟的步骤如下所示

（1）步骤一：PTP 线程向定时器线程注册定时周期，定时器线程按照 PTP 线程所注册的定时，周期性给 PTP 发送定时 msg。然后编排调度从定时器 txq 队列读取到定时的 msg 后，查表结果为 ptp 线程服务 ID，最后把 msg 发送到 PTP 线程的 rxq 队列，如上图红色标记部分。

（2）步骤二：PTP 线程接收到定时 msg，开始构建 sync 报文，然后发送 msg 到自己的 txq 队列中，编排调度读取到 PTP 线程的 txq 队列中的 msg，查表结果为数据发送线程，把 msg 发送到数据发送线程的 rxq 队列，数据发送线程次取出 rxq 队列中的 msg，并将 msg 所承载的报文发送到硬件平台。如上图蓝色标记部分。

（3）步骤三：数据接收线程接收到硬件平台发送的报文，根据报文类型生成 msg，并把 msg 写入到发送发送队列中，编排调度读取数据接收线程的发送队列，查表结果为 PTP 线程的服务 ID，把 msg

发送到 PTP 线程。如上图紫色标记部分。

从时钟的步骤与主时钟从硬件接收和发送报文的流程相同，区别是从时钟没有定时器线程参与。

3.3.4 未映射/ARP/重映射报文处理流程

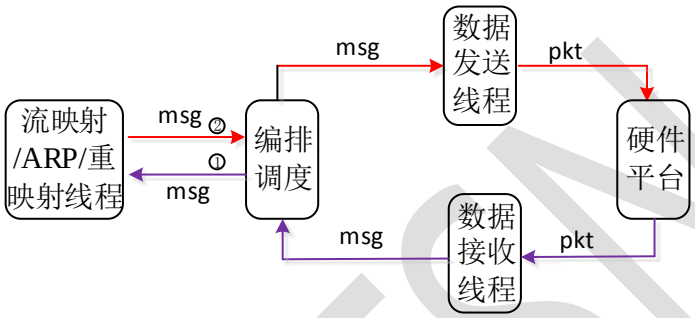


图 3-6 未映射/ARP/重映射报文处理流程图

（1）步骤一：数据接收线程接收到硬件平台发送的报文，然后根据报文类型和输入端口号（如果输入端口号为终端口，则判断是否为 ARP 报文，如果是则 msg 类型为 APR，否则为未经映射报文类型；如果输入端口不是终端口又不是已知的报文类型，则 msg 为重映射）构造 msg，写入到数据接收线程的发送队列。编排调度根据查表结果，该结果如果为流映射服务 ID，则发送到流映射线程的接收队列、如果为 ARP 服务 ID，则发送到 ARP 代理线程的接收队列、如果为重映射服务 ID，则发送到重映射线程的接收队列。如上图紫色部分所标记的。

（2）步骤二：流映射、ARP 代理和重映射线程读取各自接收队列的 msg，根据 msg 获取报文地址，进而对报文进行相应的处理，然后构造 msg，并把 msg 写入到各自的发送队列中；编排调度读取各个

队列的 msg，根据 msg 进行查表，查表结果为数据发送线程的服务 ID，则把 msg 发送到数据发送线程的接收队列；数据发送线程依次取出 rxq 队列中的 msg，并将 msg 所承载的报文发送到硬件平台。如上图红色部分所标记的。

附录 1：核心数据结构

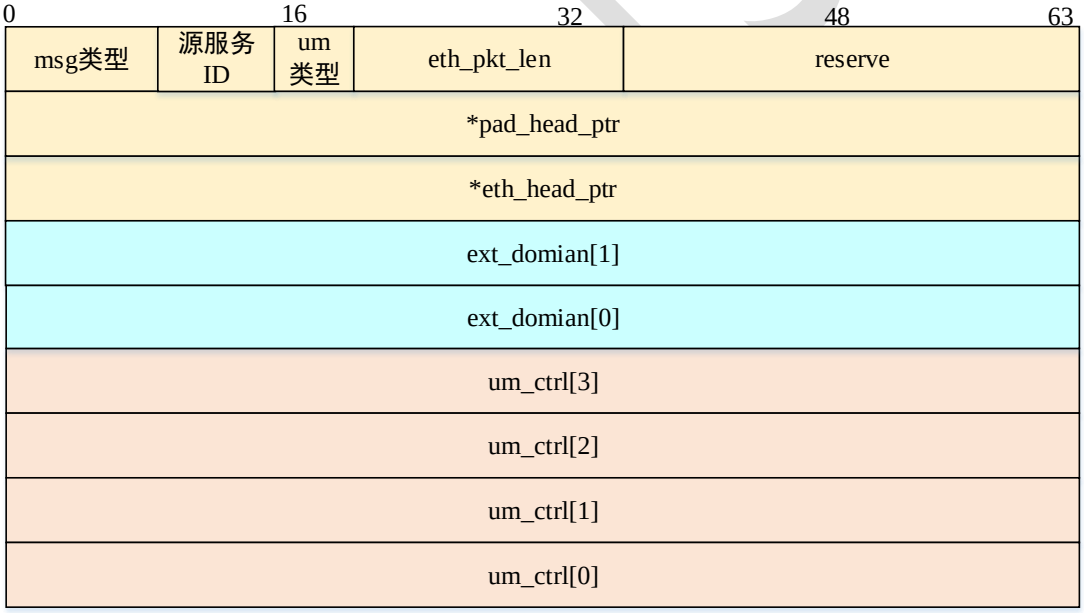
1.1 msg 定义

各线程间传递的消息内容通过 msg 结构承载，msg 似于 skb_buf 结构。

msg 总共 72 字节，前 24 字节为通用字段，适用于所有平台和所有应用。

中间的 16 字节为扩展域，允许用户根据硬件平台和应用场景需要，自定义字段以保存应用服务处理的中间结果。

最后的 32 字节拷贝数据报文的 metedata。



图附 1-1 metedata 格式

各字段的含义：

（1）msg 类型：msg 类型字段用于标识线程间所传递的消息内容所属类别，便于编排调度拼接 key。

（2）源服务 ID：标识 msg 消息上一跳处理线程的服务 ID 编号。

所有的服务线程都有一个线程编号，用于唯一标识线程的身份。msg 的前两个字段，即 msg 类型和源服务 ID，共同组成编排调度表需要的 key。

服务 ID 编号根据用户需要也可以不断扩充。

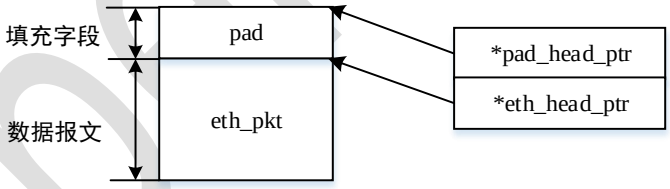
（3）um_type：metedata 的长度，主要用于标识数据报文发往硬件时 metedata 指针偏移量。

（4）eth_pkt_len：以太网报文长度。

（5）reserve：保留位；

（6）*pad_head_ptr：填充报文头地址；

（7）*eth_head_ptr：以太网报文头地址，即报文偏移 metedata 头之后的报文;



图附 1-2 报文头与以太网报文头地址

（8） ext_domian：用户自定义字段，允许用户根据硬件平台和应用场景需要自定义字段以保存应用服务处理中间的结果；

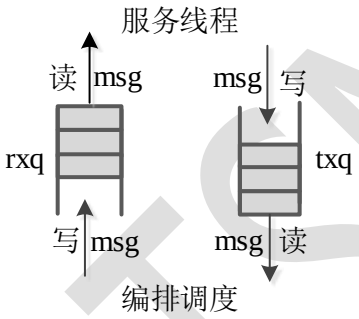
（9） um_ctrl：对于硬件送上来的数据报文有 metedata 头的情况，存放 metedata 信息。如果没有 metedata 头，则不用存放。

1.2 消息队列

各线程间传递的 msg 结构，通过消息队列存储。rxq 为接收消息队列，txq 为发送消息队列。

编排调度负责往 rxq 队列里写 msg，往 txq 队列里读 msg。

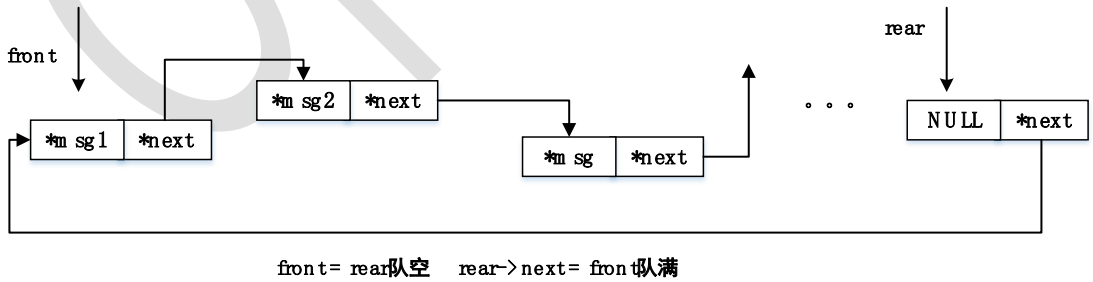
服务线程负责往 rxq 队列里读 msg，往 txq 队列里写 msg。



图附 1-3 消息队列

1.3 rxq 和 txq 定义

rxq 或 txq 消息队列为 struct msg_quene_node 类型的环形链表，如下所示。每一个队列节点都是存放 msg 指针。



图附 1-4 队列环形链表

front 和 rear 分别为队列的头指针和尾指针，用于判断队列是否满了或队列是否为空。

整个队列预留一个节点空闲。初始化时 front 和 rear 都指向第一个节点，每写入一个 msg 指针消息，rear=rear->next。每取出一个 msg 指针消息，front=front->next。

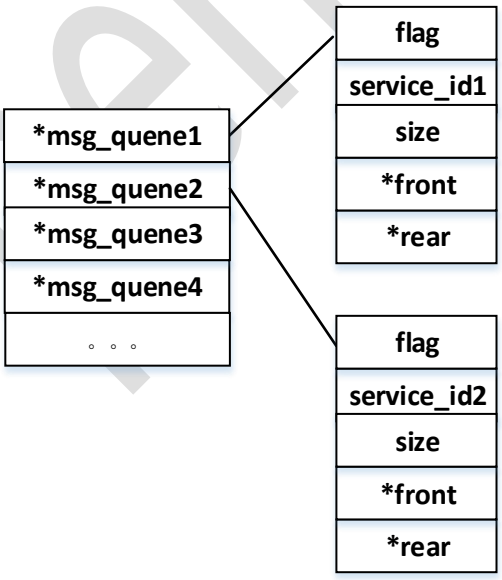
当 front = rear 时队列为空，rear->next = front 时队列满。

1.4 消息队列注册表

输入消息队列注册表 rxq_service_list 和输出消息队列注册表 txq_service_list 都是全局变量的指针数组。

线程初始化时创建自己的输入队列和输出队列并向消息队列注册表中注册。

在写入 rxq_service_list 数组前使用 rxq_mutex 加锁，写完后解锁。在写入 txq_service_list 数组前使用 txq_mutex 加锁，写完后再解锁。



图附 1-5 消息队列注册

每个队列定义了 5 个属性，分别为 flag、service_id、size、*front 和*rear。各个属于含义：

（1）flag：消息队列活跃标志，0 为不活跃，1 为活跃。每个线程把自己的消息队列注册到注册表以后，就将 flag 置为 1。主进程在编排调度的时候，通过 flag 是否为 1 来决定是否轮询相应的 txq 队列，以及是否把消息送入相应的 rxq 队列。

（2）service_id：表示哪个服务线程的消息队列，每个线程在队列初始化时将 service_id 置为自己的服务编号；

（3）size：表示消息队列的深度；

（4）*front：队列的头指针；

（5）*rear：队列的尾指针。

*front 和*rear 用于判断队列是否满了或队列是否为空。*front 每读一个消息更新值，*rear 每写一个消息更新值。