《鸿蒙操作系统分布式软总线技术》调研报告（部分）

朱浩

SA20225646

目　录

1 HarmonyOS 概述----------3
1.1 系统定义----------3
1.2 系统架构----------3
1.3 分布式技术特性----------3
2 分布式软总线模块解析----------4
2.1 分布式软总线的功能----------4
2.2 分布式软总线的原理----------4
2.3 分布式软总线源码分析----------6
2.3.1 discover 部分----------7
2.3.2 authmanager 部分----------8
2.3.3 trans_service 部分----------10
2.3.4 软总线代码执行流程----------11
3 编译、运行软总线模块----------11
4 参考资料----------11

1 HarmonyOS 概述

1.1 系统定义

HarmonyOS 是一款“面向未来”、面向全场景（移动办公、运动健康、社交通信、媒体娱乐等）的分布式操作系统。在传统的单设备系统能力的基础上，HarmonyOS 提出了基于同一套系统能力、适配多种终端形态的分布式理念，能够支持手机、平板、智能穿戴、智慧屏、车机等多种终端设备。

1.2 系统架构

HarmonyOS 整体遵从分层设计，从下向上依次为：内核层、系统服务层、框架层和应用层。系统功能按照“系统 > 子系统 > 功能/模块”逐级展开，在多设备部署场景下，支持根据实际需求裁剪某些非必要的子系统或功能/模块。HarmonyOS 技术架构如下所示。

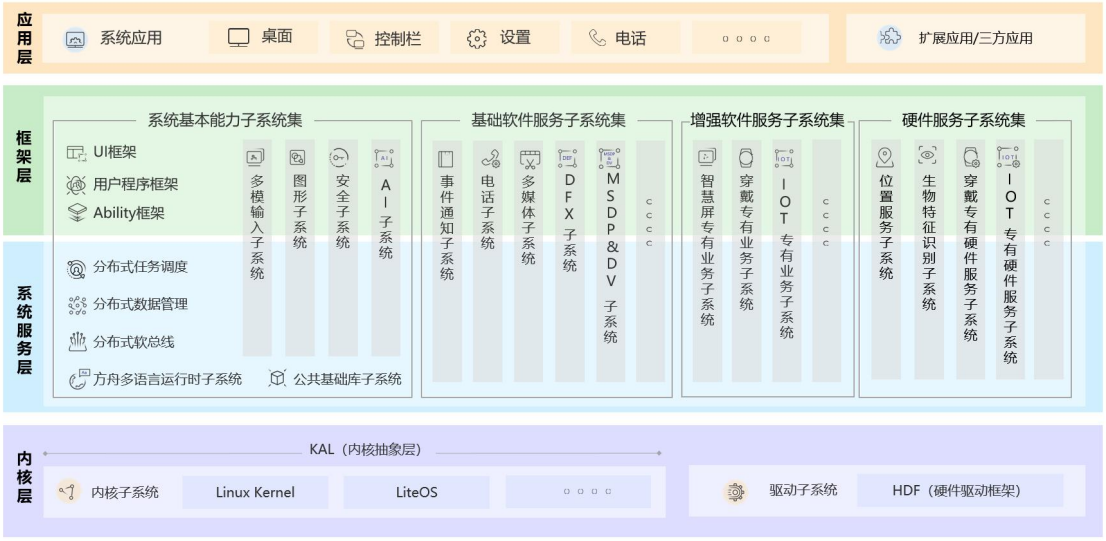


图 1：鸿蒙系统的架构层次

1.3 分布式技术特性

HarmonyOS 中，多种设备之间能够实现硬件互助、资源共享，依赖的关键技术包括分布式软总线、分布式设备虚拟化、分布式数据管理、分布式任务调度等。

图 2：鸿蒙系统所依赖的分布式技术

2 分布式软总线模块解析

2.1 分布式软总线的功能

在鸿蒙系统中，分布式软总线是手机、平板、智能穿戴、智慧屏、车机等分布式设备的通信基座，为设备之间的互联互通提供了统一的分布式通信能力，为设备之间的无感发现和零等待传输创造了条件。依托软总线技术，可以轻松实现多台设备共同协作完成一项任务，任务也可以由一台设备传递至另一台设备继续执行。对于用户而言，无需关注多台设备的组网，软总线可以实现自发现、自组网。对于开发者而言，也无需针对不同设备开发不同版本的软件、适配不同的网络协议和标准规范。

2.2 分布式软总线的原理

相较于传统计算机中的硬总线，鸿蒙系统中的分布式软总线是一条虚拟的、“无形”的总线。可以连接同处于一个局域网内部的所有鸿蒙设备（1+8+N，如下图所示），并且具有自发现、自组网、高带宽和低时延等特点。

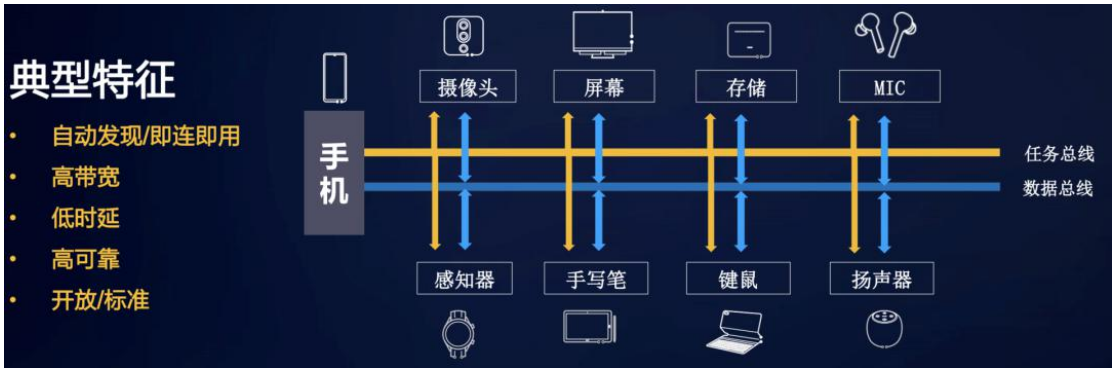


图 3：软总线连接 1+8+N 设备

除了连接处于同样网络协议中的硬件设备，软总线技术还支持对不同协议的异构网络进行组网。传统场景下，需要蓝牙传输的两台设备必须都具有蓝牙，需要 WiFi 传输的设备必须都具有 WiFi。而蓝牙/WiFi 之间是无法进行数据通信的。软总线提出蓝牙/WiFi 融合网络组网技术（架构如下图所示），解决了不同协议设备进行数据通信的问题。使得多个鸿蒙设备能够自动构建一个逻辑全连接网络，用户或者业务开发者无需关心组网方式与物理协议。

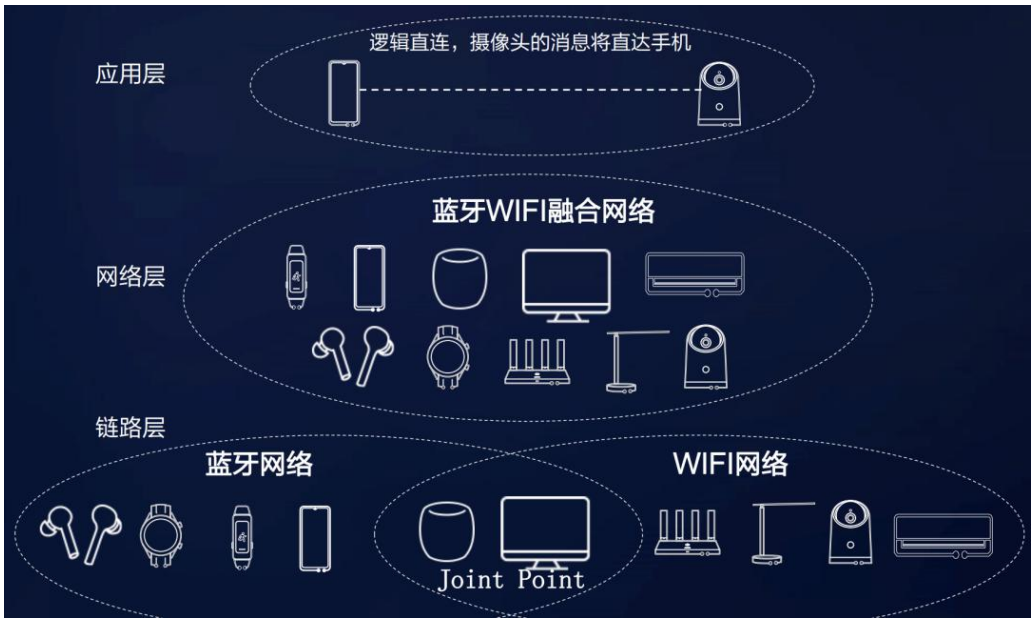


图 4：软总线中异构网络组网方案

传统协议的传输速率差异较大，多设备交互式时延和可靠性也难以保证。软总线传输提出三个目标：高带宽、低时延、高可靠。相较于传统网络的 7 层模型，软总线提出了 4 层的“极简协议”（如下图所示），将中间的 4 层协议精简为一层以提升有效载荷，有效带宽提升 20%。设备间基于 UDP 协议进行数据传输，摒弃传统滑动窗口机制，实现丢包快速回复，且具有智能网络变化感知功能，可以自适应流量控制和拥塞控制。

图 5：软总线提出“极简协议”

2.3 分布式软总线源码分析

注意：笔者所分析源码为软总线模块部分，如需获取软总线源码，请访问网址：https://gitee.com/openharmony/communication_services_softbus_lite。浏览整个软总线组件的源码，可知其分为以下 4 个单元（代码结构如下图所示）：

1，discover：提供基于 COAP 协议的设备发现机制；

2，authmanager：提供设备认证机制和知识库管理功能；

3，trans_service：提供身份验证和数据传输通道，

4，os_adapter：检测运行设备性能，决定部分功能是否执行。

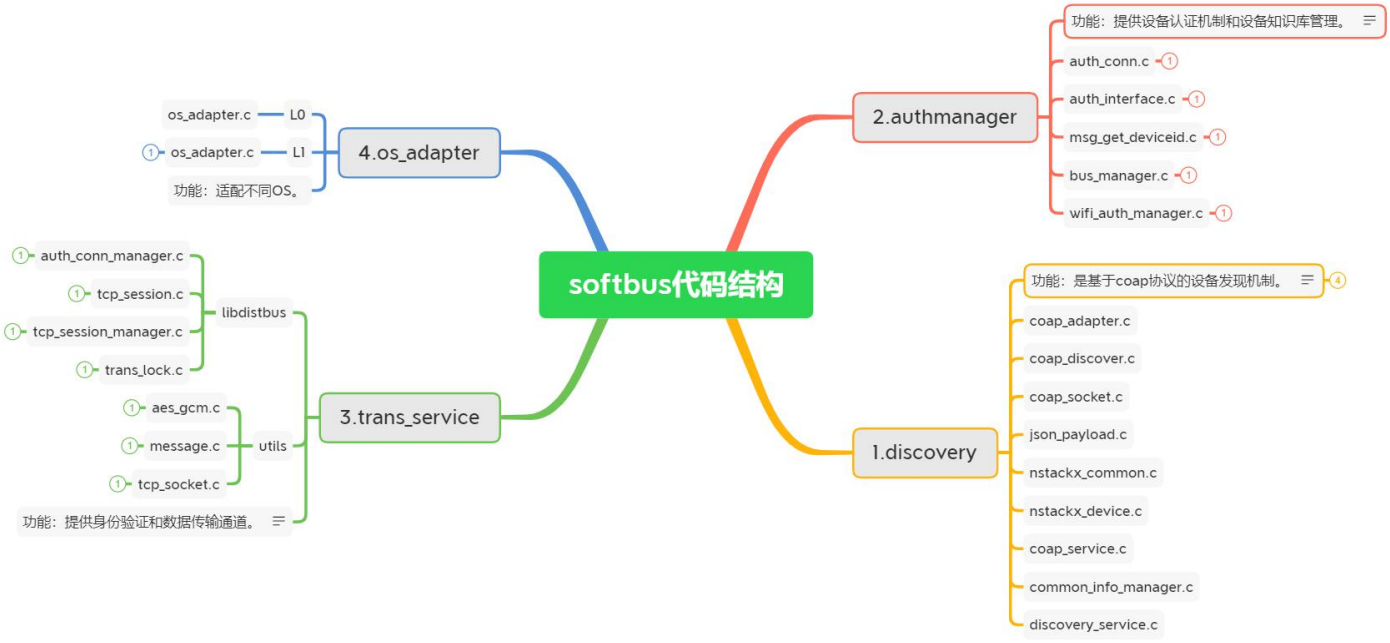


图 6：分布式软总线组件源码整体架构

由于整个软总线组件代码量较多（笔者粗略估算在 5000 行上下），为表述清晰，部分细节可能会作适当忽略（如忽略各头文件）。因 os_adapter 部分并非软总线模块的功能重点，此处将不做讨论（简要说明，os_adapter 的功能是判断当前运行设备的性能，已决定部分功能是否启用，如多线程机制）。本报告采用图文结合的形式，先对 3 个核心部分（discovery、authmanager、trans_service）的功能进行简要概述，最后将各单元聚合进行整体代码执行流程的分析。

2.3.1 discover 部分

作为鸿蒙 OS 分布式软总线重要组成单元，discovery 单元提供了基于 coap（Constrained Application Protocol，受限应用协议，RFC7252）协议的设备发现机制。为什么使用 coap 协议？是因为考虑到运行 harmonyOS 的设备除了硬件性能较好的手机、电脑等设备，还有资源受限的物联网设备，这些设备的 ram、rom 相对较小。coap 协议支持轻量的可靠传输，采用 coap 协议，可以扩大组网范围。

discovery 的实现前提是确保发现端设备与接收端设备在同一个局域网内且能互相收到对方的报文。流程为以下三步：

1，发现端设备，使用 coap 协议在局域网内发送广播；

2，接收端设备使用 PublishService 接口发布服务，接收端收到广播后，发送 coap 协议单播给发现端；

3，发现端设备收到回复单播报文,更新设备信息。

discovery 部分代码由两部分组成（目录如下图所示）。其中 coap 部分是 coap 协议的封装实现，discovery_service 是基于 coap 协议的设备间发现流程的实现。

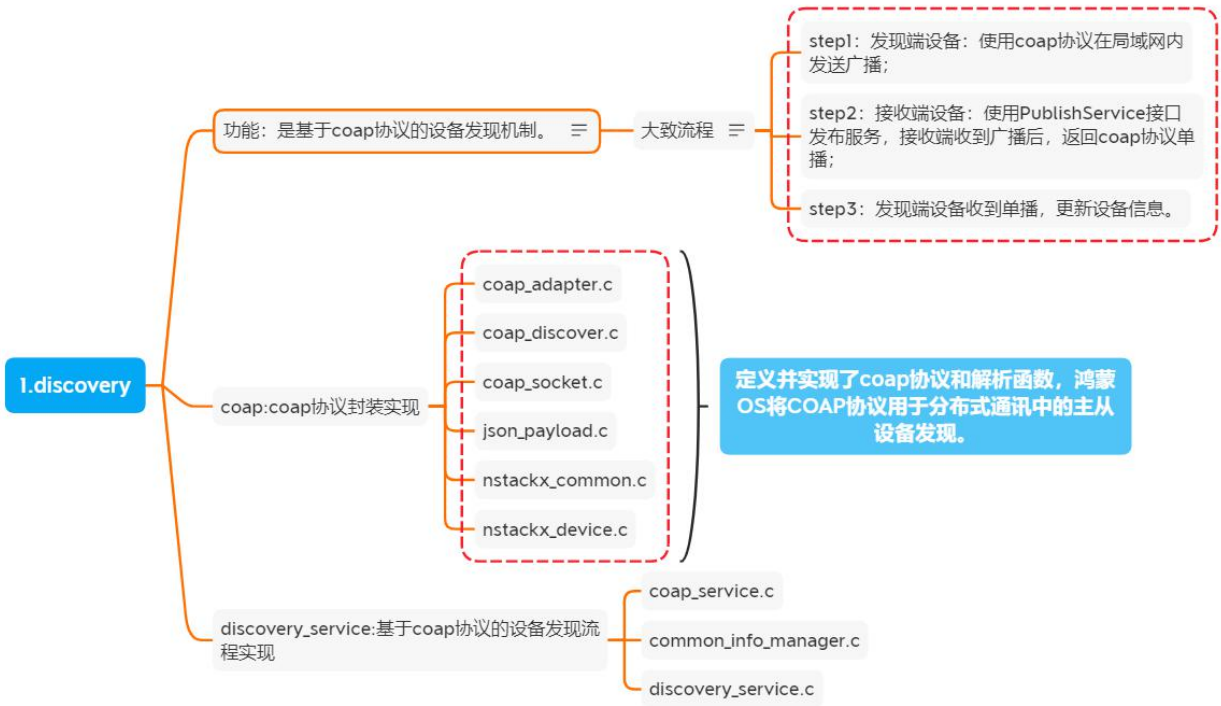


图 7：discovery 部分源码目录结构

coap 目录中：

（1）coap_def.h：定义 coap 协议包的格式、报文结构，且使用 UDP 协议传输；

（2）coap_adapter.c：实现 coap 协议的编码、解码函数；

（3）coap_socket.c：实现 coap 包的发现、接收服务；

（4）Coap_discovery.c：实现基于 coap 协议的设备发现功能。本文件定义了 socket 通讯过程，有兴趣的读者可以深入阅读。后文也会对本文件关键函数流程做分析。

discovery_service 目录中：

（5）comman_info_manager.h：定义了鸿蒙系统当前支持的设备类型与级别；

（6）Discovery_service.c：实现了设备暴露、发现和连接流程。这里需要注意的是，考虑到同一局域网下，主设备发出连接请求广播后，多个物联网设备都会回复单播应答报文从而造成信道冲突。为避免此情况发生，每个物联网设备均维护一套信号量机制，实现多设备的有序等待。

2.3.2 authmanager 部分

作为软总线代码执行流程中的第二部分：authmanager 单元提供了设备认证机制。设备通过加密和解密的方式，互相建立信任关系，确保在互联场景下，用户数据在对的设备之间进行流转，实现用户数据的加密传输。软总线中定义加密和解密算法的内容在 trans_service/utils/aes_gcm.h 中，authmanager 中的处理流程如下图所示：

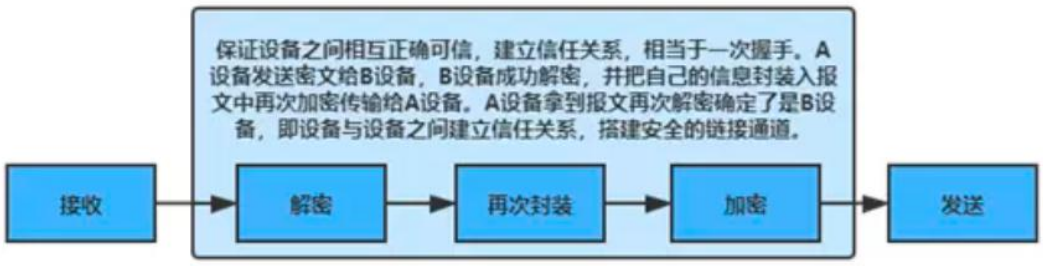


图 8：加密传输建立设备间信任关系

authmanager 单元代码目录结构如下图所示：

图 9：authmanager 代码结构

authmanager 目录中：

（1）auth_conn.c：提供发送、接收、认证和获取密钥的功能；

（2）auth_interface.c：管理会话、链接、密钥节点，提供增删改查功能；

（3）msg_get_deviceid.c：以 cJSON 格式获取各个设备的信息，包括设备 id、链接信息、设备名、设备类型等；

（4）bus_manager.c：创建不同的 listen，用以监听系统上有哪些 device 并创建新的 device 节点，以及节点数据的处理。bus_manager.c 主要由 discovery 单元调用，通过判断本文件中 flag 标志位决定是否启动总线（start_bus()函数）或关闭当前总线（stop_bus()函数）。discovery 调用后，bus_manager 执行流程如图 10：

（5）wifi_auth_manager.c：实现了链接管理和数据接收功能。

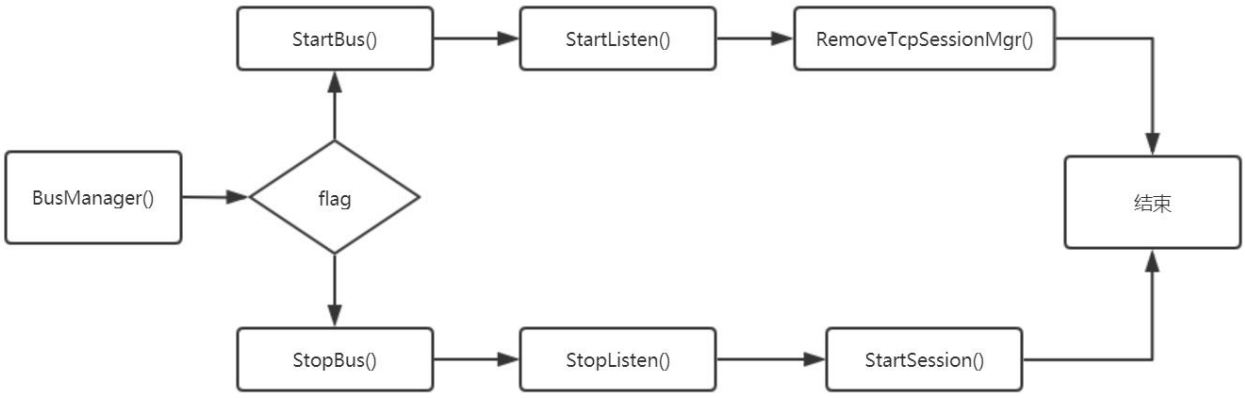


图 10：bus_manager.c 中函数执行流程图

2.3.3 trans_service 部分

经过第一阶段协议确定、设备发现，第二阶段设备链接，软总线模块执行到了第三阶段：数据传输阶段，即目录中 trans_service 单元。trans_service 模块依赖于 harmonyOS 提供的网络 socket 服务，向认证模块提供认证通道管理和认证数据的收发；向业务模块提供 session 管理和基于 session 的数据收发功能，并且通过 GCM 模块的加密功能提供收发报文的加密/解密保护。如下图所示为 trans_service 模块在系统架构中的位置：

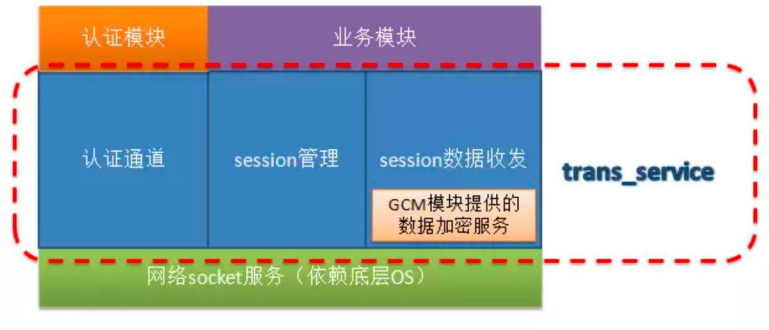


图 11：trans_service 模块在 OS 结构中的位置

阅读本单元部分源码，可知数据传输单元的源码目录如下：

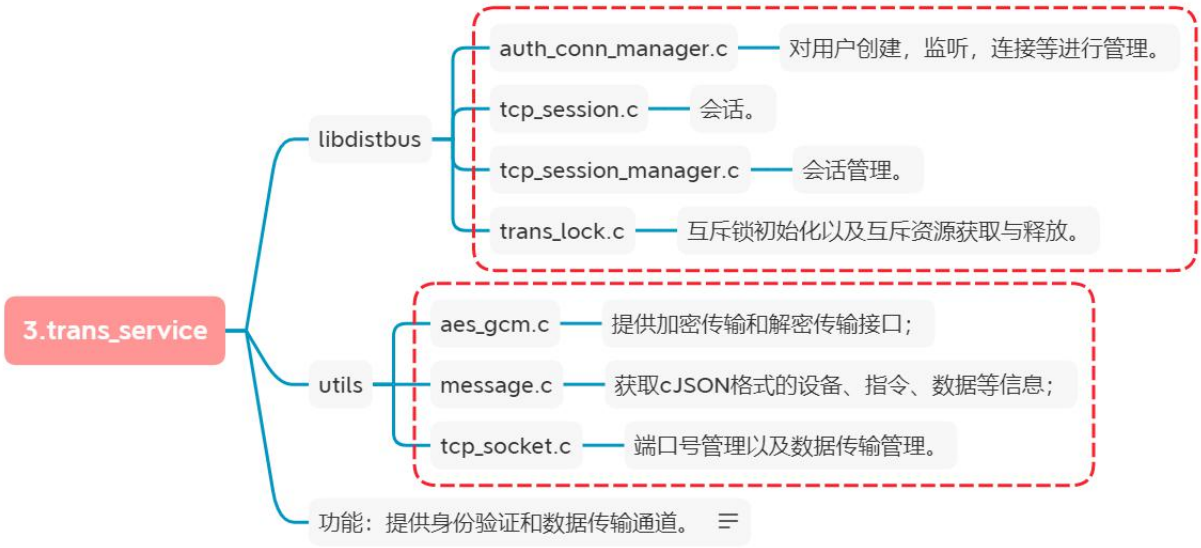


图 12：trans_service 模块源码目录

在 libdistbus 目录中，各源码文件的实现功能均已给出（如上图 12），其中需要注意的是，软总线启动后，设备在发现阶段基于 coap 协议使用 udp 数据报进行通讯。当设备认证通过，确认连接后，将会使用 TCP 协议进行更加安全可靠的通讯。同发现阶段避免多设备同时向主机回复单播报文产生信道冲突一样，这里也维护了一套

信号量机制避免多设备互连产生资源冲突。对这部分感兴趣的读者可以继续阅读 trans_lock.c 文件，这里不再赘述。

2.3.4 软总线代码执行流程

通过对 3 个核心组成部分的源码进行简单的分析，相信读者已经对软总线的具体结构和各部分功能有了一定了解。下面，我们将融合这 3 个单元，对软总线启动后的源码执行流程做进一步分介绍。

在分布式软总线的设计中，trans_service 模块是在 authmanager 模块中被初始化的，而 authmanager 模块又被 discovery 模块初始化，因此设备在向外发布本设备信息的过程中，这三个相互关联的模块会以“发现→认证→传输”的顺序完成初始化（如下图 12）。

图 12：3 个模块的激活顺序

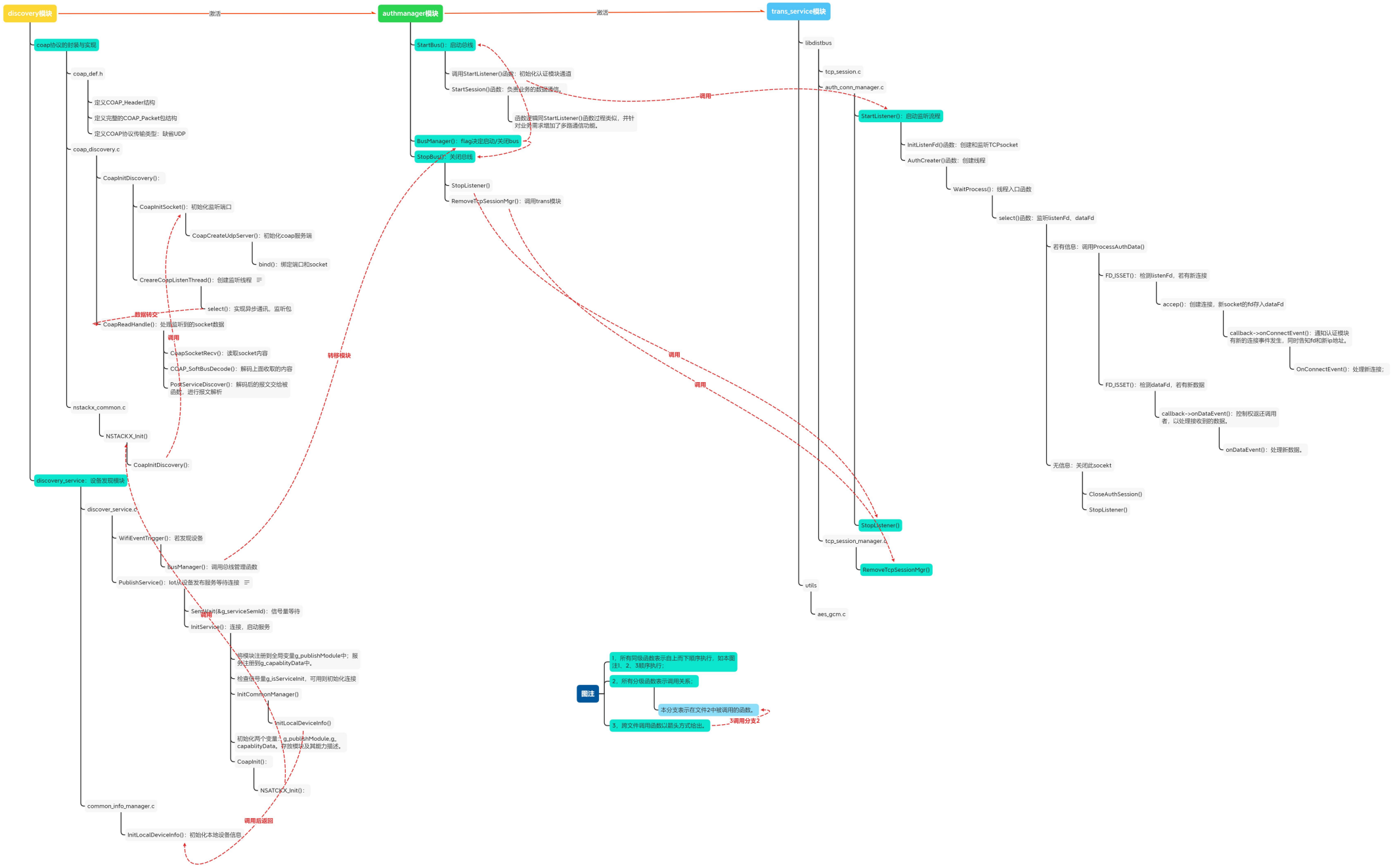
为了便于理解软总线执行流程，屏蔽具体函数细节，笔者将这一过程中的函数调用关系绘制成图（篇幅所限，图片置于文末），并标注了执行顺序，读者可以按照流程图进行阅读。源码文件数目较多，且文件之间的调用关系错综复杂，若有疏漏错误之处，望读者见谅。

3 编译、运行软总线模块（调研中）

3.1 编译分布式软总线模块

3.2 部署及运行软总线模块

4 参考资料


discovery模块
激活
authmanager模块
激活
trans_service模块
coap协议的封装与实现
coap_def.h
定义COAP_Header结构
定义完整的COAP_Packet包结构
定义COAP协议传输类型：缺省UDP
coap_discovery.c
CoapInitDiscovery()：
CoapInitSocket()：初始化监听端口
CoapCreateUdpServer()：初始化coap服务端
bind()：绑定端口和socket
CreareCoapListenThread()：创建监听线程
select()：实现异步通讯，监听包
数据转交
CoapReadHandle()：处理监听到的socket数据
调用
CoapSocketRecv()：读取socket内容
COAP_SoftBusDecode()：解码上面收取的内容
PostServiceDiscover()：解码后的报文交给被函数，进行报文解析
nstackx_common.c
NSTACKX_Init()
CoapInitDiscovery()：
discovery_service：设备发现模块
discover_service.c
WifiEventTrigger()：若发现设备
busManager()：调用总线管理函数
PublishService()：IoT从设备发布服务等待连接
SemWait(&g_serviceSemId)：信号量等待
调用
InitService()：连接，启动服务
将模块注册到全局变量g_publishModule中；服务注册到g_capabilityData中。
检查信号量g_isServiceInit，可用则初始化连接
InitCommonManager()
InitLocalDeviceInfo()
初始化两个变量：g_publishModule.g_capabilityData，存放模块及其能力描述。
CoapInit()：
NSATCKX_Init()：
common_info_manager.c
InitLocalDeviceInfo()：初始化本地设备信息
调用后返回
转移模块
StartBus()：启动总线
调用StartListener()函数：初始化认证模块通道
StartSession()函数：负责业务的数据通信。
函数逻辑同StartListener()函数过程类似，并针对业务需求增加了多路通信功能。
BusManager()：flag决定启动/关闭bus
StopBus()：关闭总线
StopListener()
RemoveTcpSessionMgr()：调用trans模块
调用
调用
libdistbus
tcp_session.c
auth_conn_manager.c
调用
StartListener()：启动监听流程
InitListenFd()函数：创建和监听TCPsocket
AuthCreater()函数：创建线程
WaitProcess()：线程入口函数
select()函数：监听listenFd、dataFd
若有信息：调用ProcessAuthData()
FD_ISSET()：检测listenFd，若有新连接
accep()：创建连接，新socket的fd存入dataFd
callback->onConnectEvent()：通知认证模块有新的连接事件发生，同时告知fd和新ip地址。
OnConnectEvent()：处理新连接；
FD_ISSET()：检测dataFd，若有新数据
callback->onDataEvent()：控制权返还调用者，以处理接收到的数据。
onDataEvent()：处理新数据。
无信息：关闭此socekt
CloseAuthSession()
StopListener()
StopListener()
tcp_session_manager.c
RemoveTcpSessionMgr()
utils
aes_gcm.c
图注
1，所有同级函数表示自上而下顺序执行，如本图注1、2、3顺序执行；
2，所有分级函数表示调用关系；
本分支表示在文件2中被调用的函数。
3，跨文件调用函数以箭头方式给出。
3调用分支2