ICS 35.160

CCS C 39

团 体 标 准

T/GCC 3007—2025

统一基本输入输出系统（UBIOS）

基础架构规范

Unified Basic Input/Output System (UBIOS) Base Architecture Specification

2025-10-17 发布

2025-10-17 实施

全球计算联盟 发布

目　　次

前　言

本文件按照GB/T 1.1-2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由全球计算联盟新一代BIOS专业委员会提出。

本文件由全球计算联盟归口。

本文件起草单位：中国电子技术标准化研究院、华为技术有限公司、南京百敖软件有限公司、昆仑太科(北京)技术股份有限公司、超聚变数字技术有限公司、四川华鲲振宇智能科技有限责任公司、天翼云科技有限公司、统信软件技术有限公司、河南昆仑技术有限公司、软通计算机有限公司、神州数码集团股份有限公司、中移(苏州）软件技术有限公司、宝德计算机系统股份有限公司。

本文件主要起草人：马银川、宋东匡、聂永丰、王辉强、陈刚、包振忠、王东伟、尹航、钟伟军、李雪莲、吴平、高黎明、陈小春、张家定、高鹏飞、陈刚、赵彦钧、刘东、季良杰、刘宇、李萌、牛战争、王昱力、徐志刚、李亚军、邓忠良、彭小南、夏军、王晓辉、刘敬轩、龚焕新、潘晓信。

统一基本输入输出系统（UBIOS）基础架构规范

1 范围

本文规定了计算系统中基本输入输出系统（BIOS）、操作系统（OS）、基板管理控制器（BMC）、外设板卡等各组件间以及BIOS内部固件间的交互方式，包括交互架构、交互通道、接口形式等。

本文件适用于BIOS、操作系统、BMC和外设的设计、开发和测评。

2 规范性引用文件

本文件中没有规范性引用文件。

3 术语和定义

下列术语定义适用于本文件。

3.1 统一基本输入输出系统（Unified Basic Input Output System, 简称 UBIOS）

一种支持分布式架构与软硬芯协同的固件框架。

3.2 UBIOS 接口（UBIOS Interface）

每个组件对外提供的UBIOS接口以及如何使用其他组件提供的UBIOS接口，包括功能调用、信息传递等。

3.3 统一虚拟总线（Unified Virtual Bus，简称 UVB）

通过软件抽象屏蔽各种底层硬件通信通道，为使用者提供统一的操作接口，实现组件间互相通信的软件模拟的总线。

3.4 功能标识（Function ID，简称 FID）

表示某种功能的ID，独立定义，与模块或子模块ID解耦，由12bits描述。

3.5 信息标识（Information ID，简称 IID）

表示某种信息的 ID，独立定义，与模块或子模块 ID 解耦，由 12bits 描述。

3.6 调用标识服务（Call ID Service，简称 CIS）

指以 Call ID 为标识的一系列接口集合，每个 Call ID 对应一个功能。

3.7 通知标识信息（Notify ID Information，简称 NII）

指将信息主动上报的系列接口集合，每一个 Notify ID 代表一种确定的上报的信息内容。

3.8 调用标识服务信息表（Call ID Service Table）

用于Call ID Service提供者向调用者告知支持的Call ID Service范围及调用方式的信息表。

3.9 组件标识（User ID）

可以是Sender User ID，也可以是Receiver User ID，或者是Responser User ID。

3.10 UB 处理单元（UB Processing Unit，简称 UBPU）

支持 UB 协议栈的处理单元，实现特定功能。

3.11 实体（Entity）

设备分配其自身资源的基本单元，每个 Entity 都是 UB域内的一个通信对象。

4 缩略语/Abbreviation

下列缩略语适用于本文件。

ARM	高级精简指令集计算机器	（Advanced RISC Machine）
ATF	ARM 可信固件	（ARM Trusted Firmware）
BIOS	基本输入输出系统	（Basic Input Output System）
BMC	基板管理控制器	（Baseboard Management Controller）
CPU	中央处理单元	（Central Processing Unit）
DDR	双倍数据速率内存	（Double Data Rate）
DMA	直接内存访问	（Direct Memory Access）
EC	嵌入式控制器	（Embedded Controller）
EID	扩展号	（Extension ID）
EL	异常等级	（Exception Level）
FID	函数号	（Function ID）
GIC	通用中断控制器	（General Interrupt Controller）
HBM	高带宽内存	（High Bandwidth Memory）
IPMI	智能平台管理接口	（Intelligent Platform Management Interface）

ISA	指令集架构	（Instruction Set Architecture）
KB	千字节	（Kilobyte）
OS	操作系统	（Operation System）
RAM	随机存取存储器	（Random Access Memory）
RISC	精简指令集计算机	（Reduced Instruction Set Computer）
SoC	系统级芯片	（System on Chip）
SRAM	静态随机存取存储器	（Static Random-Access Memory）
TTLV	标识类型长度值	（Tag-Type-Length-Value）

5 概述

本文件是为适应芯片异构化、BIOS多固件化、外设智能化而设计，统一了BIOS与OS之间、BIOS与外设之间以及BIOS内部固件间的交互方式，进而延伸到与BIOS关联的其他部件，如BMC固件、EC固件。对于主流的异构SoC，典型系统分层架构见1.1.1.1.1 Step 11.图 1。

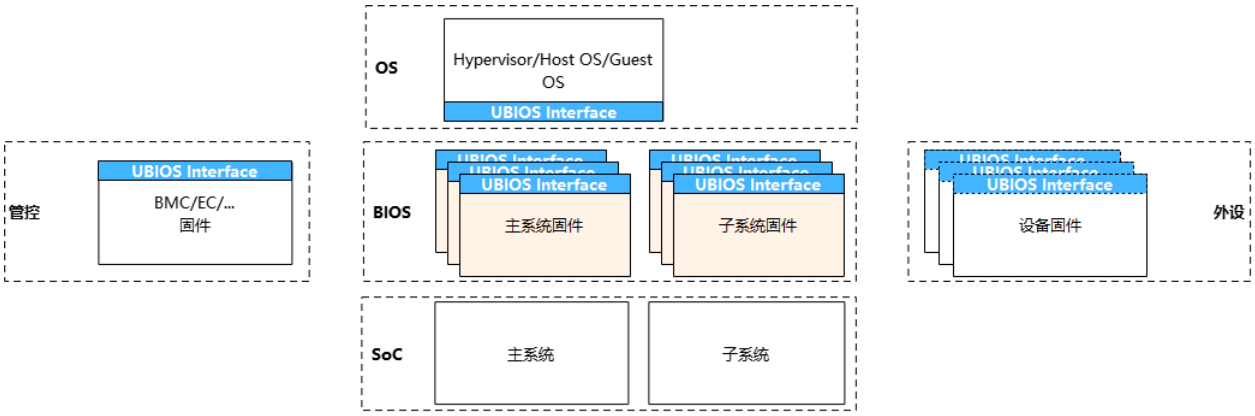

图 1 典型系统分层架构

a) 主系统：指用于计算的主系统，OS（泛指操作系统，包括Hypervisor、Host OS、Guest OS等，下同）运行在主系统上。
b) 子系统：指主系统之外，具有独立取指令和执行指令能力的计算单元，通常为实现特定功能而设计。
c) 主系统固件：运行在主系统上的BIOS固件。
d) 子系统固件：运行在子系统上的BIOS固件。
e) BMC/EC/... 固件：运行在BMC/EC等单板管理芯片上的固件或软件。
f) 设备固件：运行在外设上的固件或软件。

注：

a) 外设的 UBIOS Interface 为虚线表示依赖外设具备独立的程序执行单元，能够处理和响应 UBIOS Interface 所定义的功能。
b) 本文件与芯片架构松耦合，可应用于不同的处理器架构，包括ARM、RISC-V等。
c) 无特别说明时，本文件所有数据采用小端模式；
d) 无特殊说明时，本文件所有地址都是系统物理地址；

6 核心架构

6.1 概述

UBIOS基础架构包括虚拟总线以及通过虚拟总线传输信息的UBIOS接口。UBIOS接口包括BIOS启动OS上报信息表、BIOS的运行时服务功能、BIOS根据事件触发向OS报告的信息、 BIOS内部多组件间的功能调用和信息报告、BIOS与其他组件间的功能调用和信息报告等。UBIOS接口可分为信息上报和功能交互两类。

6.2 虚拟总线

本文件提出了统一虚拟总线（UVB）的概念，以简化 BIOS 内部多固件及 BIOS 与周边组件之间的复杂交互，使各组件都可基于相同的接口定义进行交互。

UVB 是面向各种组件提供统一接口的、虚拟的总线。它可以通过不同的物理实体承载，SoC 内所有子系统都可连接到这个总线上，SoC 外围部件也可与 UVB 相连。交互消息可通过总线传递，以达到统一各组件间的软件交互方式的目的，见图 2。

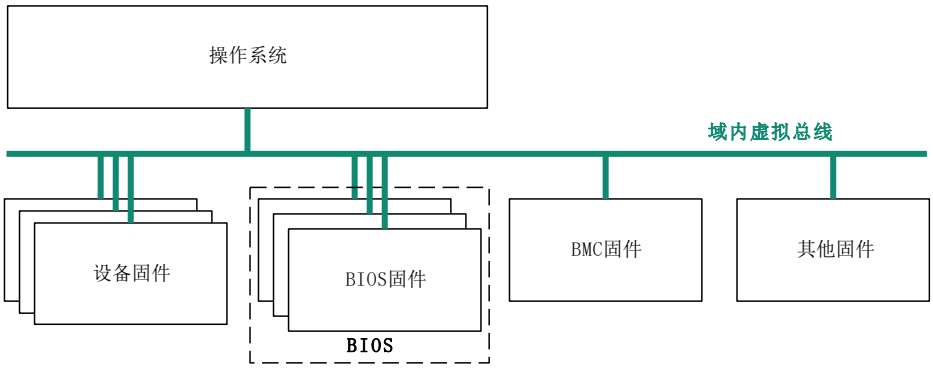

图 2 虚拟总线

另外，在一个大型计算系统内可能存在多个小型计算系统，各小型计算系统通过互联总线连接，每个小型计算系统都拥有自己的 SoC 芯片、BIOS、OS，这种情况在本文件中称为多域系统。这个系统可以是对称的，即每个域拥有相同的硬件环境、相同的 BIOS、相同的 OS 等；也可以是非对称的，即各个域的 BIOS、OS 可以不同，常见于异构计算集群中。不论对称还是非对称的多域系统，均可通过 UVB 互联。见图 3。

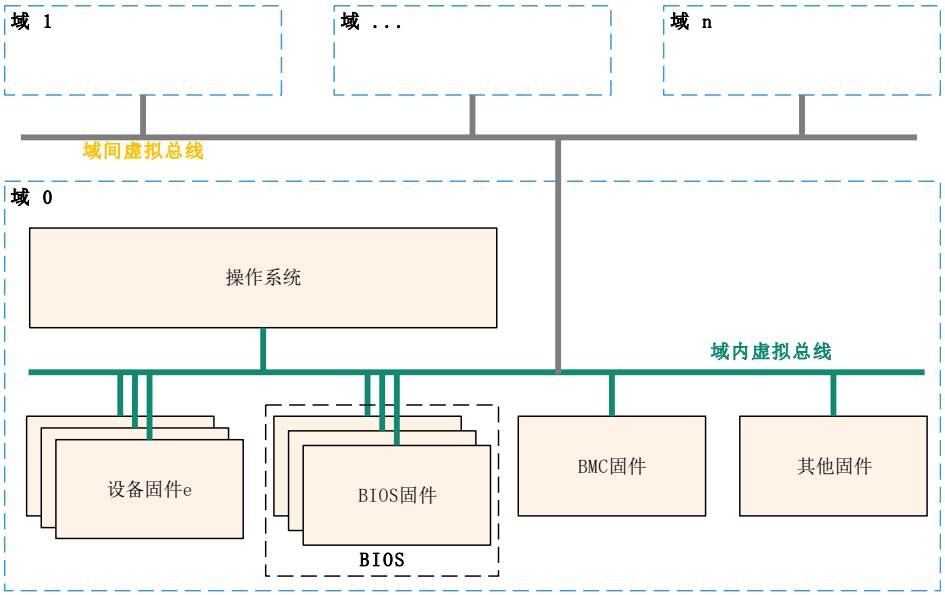

图 3　域间虚拟总线

注：域内虚拟总线和域间虚拟总线支持同一条总线，也支持子母总线分级实现，即每个域内独立设置一条虚拟总线，内部通信忽略域的概念。域与域之间设置一条虚拟总线来进行域间交互。

虚拟总线的典型特征：

a) 虚拟总线面向功能编程、接口统一，跨组件交互时只需关注功能，无需关注物理通道；

b) UVB通过统一的接口抽象不同的物理通道。组件的扩展，只需接入UVB即可与UVB上其他组件进行交互；

c) 组件间通过UVB进行信息传递，并在存在安全访问限制的场景中设置专用的缓存区域，通信双方只允许读写这一专用缓存，并通过UVB本身机制进行安全防护；

d) 各组件的实际交互对象是UVB，而不是某个具体组件， UVB提供的接口并不随硬件变化而改变，具有兼容性。

6.3　交互通道

交互通道即 BIOS 与周边组件之间以及 BIOS 内部各组件之间进行信息传输的通道。交互通道按照所处位置可分为 SoC 内和 SoC 外，按照信息传递方向可分为垂直方向和水平方向， 见图 4。

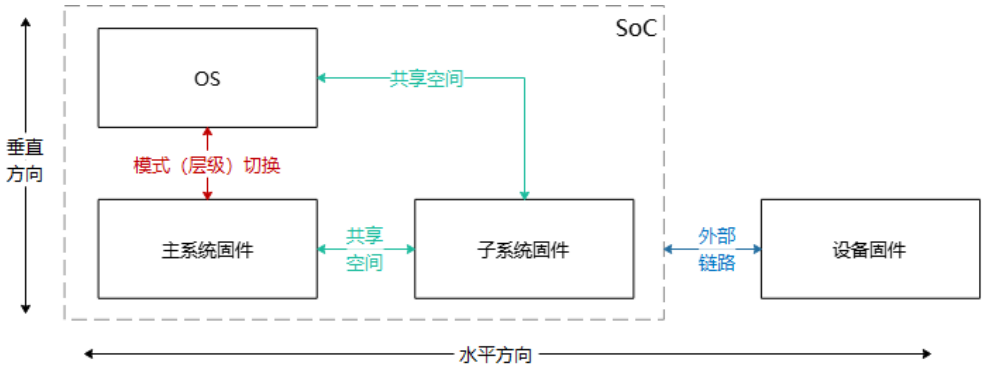

图 4　典型的交互通道

SoC 内的交互通道包含垂直交互通道和水平交互通道。

SoC 内固件与 OS 之间的信息交互通道属于垂直交互通道，通过芯片体系结构规定的芯片专用指令进行信息交互。

示例：比如 ARM64 的 Exception Level（EL），其中 EL3 属于 BIOS 层，EL2、EL1、EL0 属于 OS 层，EL 之间通过专门定义的指令进行切换，并通过共享的通用寄存器进行信息传递，这种通过 ISA 指令完成信息传递的通道即为芯片 ISA 通道。

水平交互通道，即共享空间通道，可支持跨芯片体系架构的信息交互，此时通信双方均能访问彼此约定的某处内存区域。

SoC 外只有水平交互通道，通过其他协议实现信息交互，具体协议由硬件链路决定。

6.3.1 芯片 ISA 通道

本文件芯片 ISA 交互仅涉及与 BIOS 交互所需的芯片 ISA 特性，指可以将 CPU 控制权从 OS 切换到 BIOS 的指令或指令组，如 ARM 架构的 SMC（Secure Monitor Call，ARM 架构中用于在不同安全状态之间进行切换的指令）、RISC-V 架构的 Ecall（Environment Call，RISC-V 架构中用于触发系统调用的指令），以及用于传递数据的通用寄存器。

注：芯片 ISA 特性跟处理器架构强相关，若下述说明与对应处理器架构手册存在冲突，以处理器架构手册说明为准。

ARM ISA指令见表 1。

表 1 ARM ISA 指令

指令	输入	输出
SMC (In AArch64)	W0 : 功能标识符 X1 ~ X6 : 输入参数	X0 ~ X3 : 输出参数
SMC (In AArch32)	R0 : 功能标识符 R1 ~ R6 : 输入参数	R0 ~ R3 : 输出参数

RISC-V ISA指令见表 2。

表 2 RISC-V ISA 指令

指令	输入	输出
ECALL	x17(a7) : EID x16(a6) : FID x10(a0) ~ x13(a5) : 输入参数	x10(a0) : 错误代码 x11(a1) : 输出结果

6.3.2 共享空间通道

共享空间是一块通信双方均可访问但没有格式定义的内存区域。在本文件中将基于共享空间创建的UVB实体称为内存虚拟总线，简称为虚拟总线（UVB），即在虚拟总线（UVB）与多种物理通道并存时，其含义特指内存虚拟总线，否则泛指虚拟总线整体架构。

6.3.3 其他协议

其他协议指SoC与外部组件间通信协议，如BIOS与BMC的通信协议，此类通信协议并不由本文件定义，但其数据段可用于传递本文件定义的各种数据。

6.4 接口形式

6.4.1 概述

接口形式即UBIOS中功能接口层所约定的接口形式或种类。

本文件将以BIOS、OS、外设及BMC等组件为边界进行接口定义，要求各组件按规范提供相应接口，接口功能具体由组件内单个固件独立完成，或者多固件联合完成，由组件开发者决定。组件内各固件的接口形式应当遵循本文件要求，与本文件定义保持一致。

按照接口使用方式，可分被动接受功能调用的接口与主动上报信息的接口两类：

a) 被动接受功能调用的接口：统一采用Call ID的方式定义，一个Call ID对应一个功能或服务，称为Call ID Service（CIS），Call ID Service是请求（request）、响应（response）的模型，响应是必备的。

b) 主动上报信息的接口：统一采用Notify ID的方式定义，一个Notify ID对应一个信息，称为Notify ID Information（NII），Notify ID Information是报告（report）、确认（ack）的模型，确认是可选的。

6.4.2 调用标识服务（Call ID Service）

Call ID Service 指以 Call ID 为标识的一系列接口集合，每个 Call ID 对应一个功能。

在多固件的 BIOS 系统中，开发者可按需将多个 Call ID Service 部署在不同固件中。本文件对 Call ID Service 的部署实现不做要求，但在一定范围内一个 Call ID Service 只允许被一个固件响应。

注： 一定范围指可以通过 User ID 来区分的范围，即 User ID 唯一。如果一个 User ID 对应多个固件，那么将会由首先获取到该命令的固件来响应。

Call ID Service 调用方式有以下 3 类：

a) 芯片ISA调用：通过通用寄存器进行Call ID Service消息传递。

b) UVB消息：各子系统之间通过UVB进行Call ID Service消息通信，进而实现Call ID Service调用。

c) 其他协议消息：通过其他通信协议传递Call ID Service信息来实现对Call ID Service的调用。

注：上述调用方式分类中未包含中断，因为上述分类主要区分 Call ID Service 的消息传递通道，中断方式既可以采用通用寄存器传递，也可以通过 UVB 传递，如果采用通用寄存器传递，则可归入芯片 ISA 调用方式，如果采用 UVB 传递，可归入 UVB 消息方式，因此没有将中断作为一种独立的调用方式来描述。

单域异构 SoC 系统中，Call ID Service 调用逻辑框图见图 5，图中每一个组件或固件都是处于运行态或运行态可触发态运行：

注：运行态可触发运行指平时没有处于运行态，但外界可以通过特定信号将它唤醒运行，这里指特权级固件。

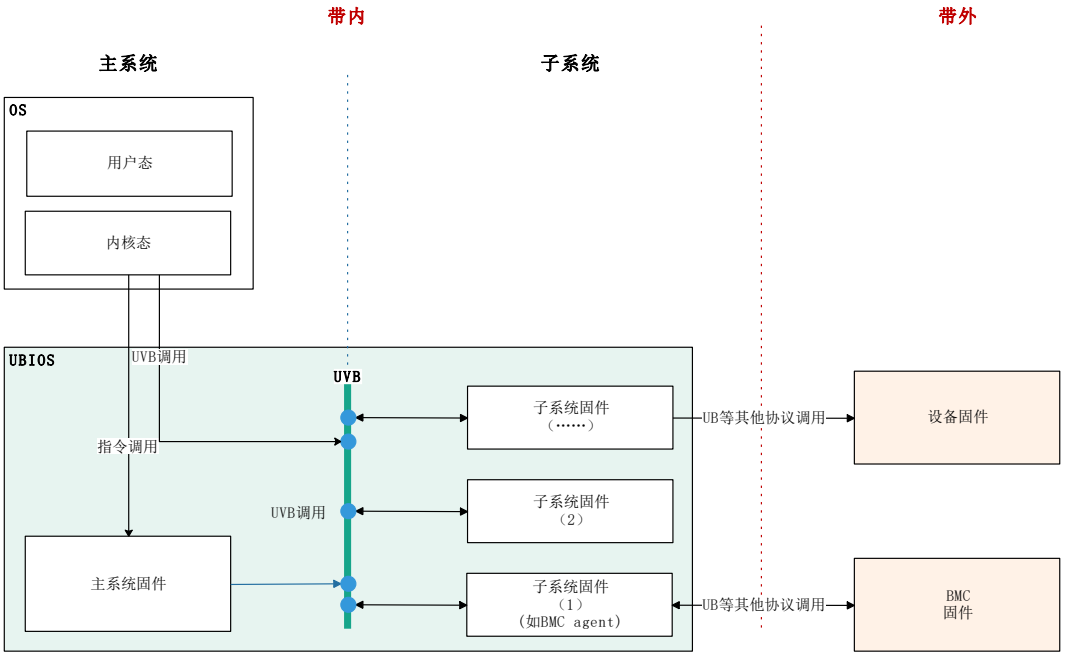

图 5 Call ID Service 调用逻辑图

Call ID Service 调用逻辑说明如下：

a) OS可通过芯片ISA调用主系统固件接口，也可通过UVB调用子系统固件的接口。

b) 主系统固件可通过UVB调用子系统固件提供的接口。

c) 子系统固件之间可通过UVB调用彼此提供的接口。

d) BMC可以通过其他协议直接或间接调用BIOS各子系统固件提供的接口，BIOS也可以反向调用BMC提供的接口。

e) 主系统固件、子系统固件、BMC可以直接或间接调用设备固件提供的接口。

在不增加额外安全手段的情况下，OS只有Kernel及更高权限的层级可以调用Call ID Service。不同实现中，同一个Call ID Service的调用方式可以不同。BIOS Call ID Service具体调用路径需要通过UBIOS信息表上报给OS。BIOS与BMC、BIOS与外设的接口由具体硬件链路决定，可不依赖UBIOS信息表（UBIOS信息表在后续章节定义）。

6.4.3 通知标识信息（Notify ID Information）

当 BIOS 检测到硬件故障或外设固件检测到外设故障时应通过 Notify ID Information 的方式主动上报其故障信息。一个 Notify ID 代表了一种上报的信息内容，不随上报主体、上报方式的不同而发生变化。

Notify ID Information 的上报应通过 UVB 消息或其他协议消息实现，要求如下：

a) UVB 消息：BIOS 准备好 Notify ID Information，并发消息到 UVB，等待 OS 发现、处理；

注：如果消息需要被及时处理，在完成 UVB 消息发送后，可通过 Notify Interrupt 通知，通知 OS 存在有效的 Notify ID Information，适用于需及时处理的场景，在消息不紧急的场景，也可以等待 OS 轮询发现。

b) 其他协议消息：

1) BIOS 向其他组件报告的场景：BIOS 准备好 Notify ID Information，并通过这类协议对外发送；

2) 其他组件（如外设）向 BIOS 报告的场景：其他组件通过这类协议向 BIOS 发送包含 Notify ID Information 的数据，由 BIOS 收集、分析，并按需向其他组件（如 OS、BMC）报告。

单域异构 SoC 系统 Notify ID Information 上报逻辑框图见图 6。

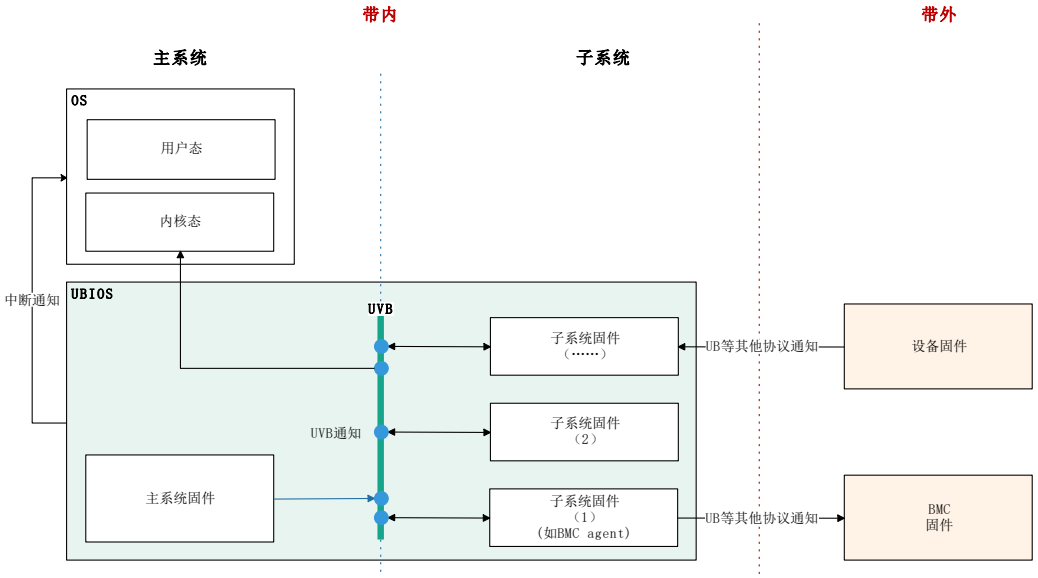

图 6 Notify ID Information 上报逻辑图

NII上报逻辑说明如下：

a) BIOS（含主系统固件和子系统固件）可以通过UVB向OS发送Notify ID Information，并且可以通过Notify Interrupt提醒OS。

b) 子系统固件向BMC发送通知消息有两种途径，一种是直接通过其他协议向BMC固件发送Notify ID Information，另一种是通过UVB借助其他子系统固件向BMC固件发送。

c) 设备固件可以通过其他协议向子系统固件发送Notify ID Information，并且可以通过子系统固件间接向OS、BMC Firmware发送。

注：

a) BIOS与OS之间需事先预留可共同访问的存储空间，用于存放Notify ID Information，地址空间在UBIOS信息表中描述，数据格式另行定义。

b) Call ID Service的响应信息是特殊的Notify ID Information，此类信息存储空间由调用者调用Call ID Service时指定，可不占用预留共享空间。

6.5 组件内分层

从组件间交互协议角度看，组件内 UBIOS 接口设计分成 2 层，信息交互层和功能接口层，支持本文件的组件均应实现这 2 层设计，见图 7。

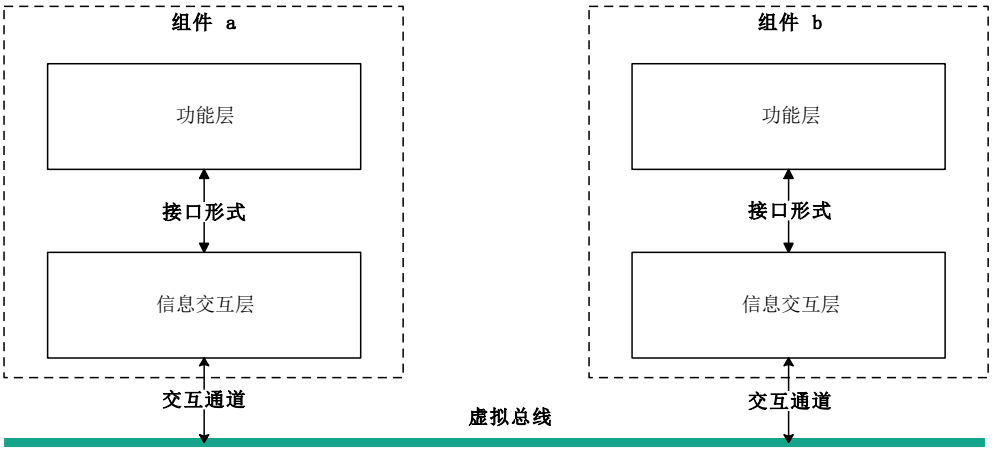

图 7 组件内分层结构

a) 信息交互层：是虚拟总线在组件内的代理，负责处理各组件间复杂的交互关系，提供统一的信息交互接口给功能接口层使用，使功能接口层可以不关心底层信息交互通道，更加聚焦在功能的实现上，功能包括但不限于：

1) 交互通道的管理和选择；
2) 功能调用的分发（回调功能接口层）；
3) 不同交互通道数据格式差异的处理；
4) 数据的有效性校验；
5) 代理与转发。

注：各组件的信息交互层在代理实施虚拟总线交互时，需要系统内具备唯一的身份（User ID），每个组件都可以是虚拟总线的使用者。

b) 功能接口层：组件接口功能集，采用一种统一的功能接口形式。

举例说明组件A调用组件B某个功能接口的详细流程，见图 8。

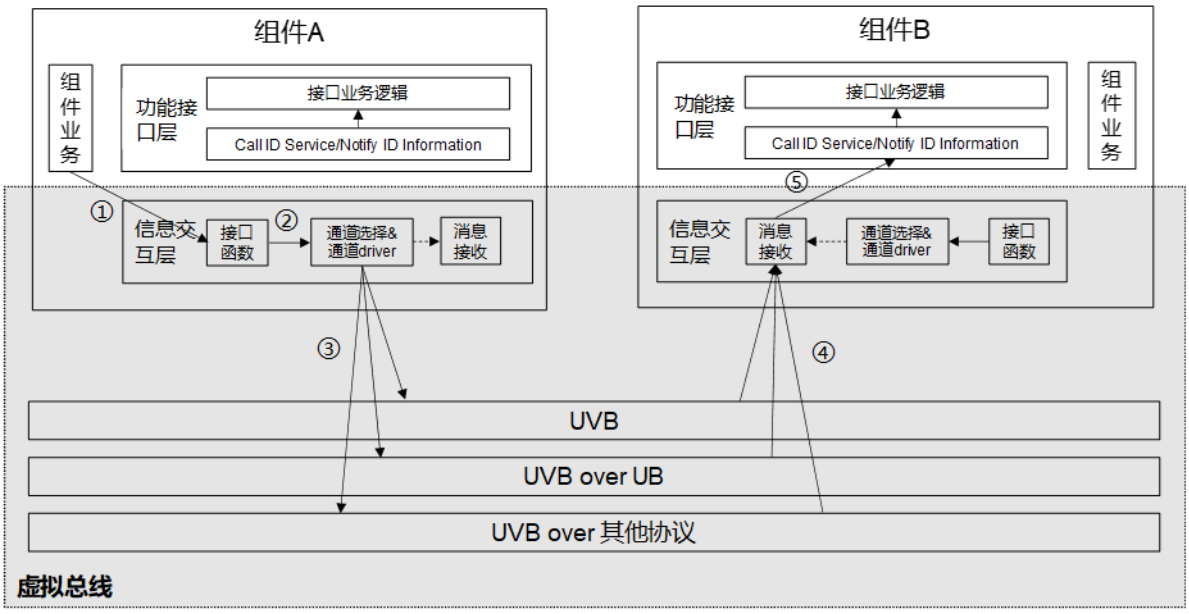

图 8 组件间调用工作流示意图

组件间调用工作步骤说明如下：

① 组件 A 组件业务逻辑中调用信息交互层接口函数，并将 Call ID，组件 B 的 User ID 及参数传递给接口函数。
② 接口函数根据配置进行通道选择。
③ 组件A信息交互层将业务调用信息传输给指定通道。
④ 组件B的消息接受模块通过交互通道获取业务信息。
⑤ 组件B信息交互层将业务信息转换成组件B可识别的信息，并将信息提供给功能接口层，完成指定功能接口调用。执行完调用后，原路径返回执行结果。

7 接口格式定义

7.1 UVB 消息标识定义

Message ID 为 UVB 消息中的消息 ID，采用 32bits 表示。内容包含为 3 个域段：Flag、Module ID、Function ID/Information ID。Message 可分为 4 类：

a) Call ID：由 Flag、Module ID、Function ID 组成。
b) Response ID：与 Call ID 一一对应，等于 Call ID 的反码。
c) Notify ID：由 Flag、Module ID、Information ID 组成。
d) Notify Ack ID：与 Notify ID 一一对应，等于 Notify ID 的反码。

Message ID 格式定义见表 3。

表 3 Message ID 格式定义

bits [31:30]	bits [29:28]	bits [27:12]	bits [11:0]
Flag	Reserved	Module ID	Function ID or Information ID

a) Flag：2bits
11b（0x3）：表示本 Message ID 是 UBIOS Call ID
10b（0x2）：用作 UBIOS Notify ID Information 的 Ack
01b（0x1）：表示本 Message ID 是 UBIOS Notify ID
00b（0x0）：用作 UBIOS Call ID Service 的 Response
b) Reserved：2bits，保留，为0
c) Module ID：16bits，表示模块分类，详细描述见Module ID内容格式章节
0x0000～0xEFFF - 本文件约定的标准模块 ID
0xF000～0xFFFF - OEM 模块 ID，预留给 OEM 厂商扩展

注：Module 指存在明确职能和功能边界的单元，有物理存在和虚拟的两种形态。

d) Function ID or Information ID：12bits，代表特定具体功能或信息，独立定义，与模块或子模块ID解耦。其中bit 11为OEM标志位，为1表示OEM自定义功能，在本文件中不对其进行规范。Function ID/Information ID要求如下：
1) Function ID和Information ID分属2个空间，分别独立编号，编号可以相同。
2) Function ID按功能类别进行分类：
0x000～0x0FF - 系统功能
0x100～0x1FF - 电源管理功能
0x200～0x2FF - RAS 功能
0x300～0x3FF - 安全功能
0x400～0x7FF - 保留

0x800～0xFFF - OEM 功能

3) Information ID按需设定，不进行分类。

0x000～0x7FF - 标准信息

0x800～0xFFF - OEM 信息

7.1.1 Function ID 模型定义

每一个 Function ID 代表一个功能，每一个功能应清晰描述以下内容：

a) Function ID：符合Function ID格式定义、无重复的ID，以十六进制格式书写。

b) 功能描述：对功能进行清晰无歧义的说明。

c) 接口类型：说明接口类型，同步功能、异步功能二选一。

注 1：同步功能：指响应者执行完成后才返回响应结果，即以 Message ID 定义中 Flag = 0x0 返回执行结果；

注 2：异步功能：响应者收到请求后先返回响应结果，再执行命令，完成执行后再通过额外的 Notify ID Information 上报，即先以 Message ID 定义中 Flag = 0x0 返回已接受命令，执行后再以 Flag = 0x1 返回执行结果。

d) 输入：描述调用此功能时需输入的参数，包括参数格式、参数含义等，要求数据格式自对齐，不依赖运行环境。

e) 输出：描述功能执行后的输出信息，包括数据格式及每个数据的含义，要求数据格式自对齐，不依赖运行环境。

f) 返回值：即执行状态，表示此次Function执行成功、失败等状态。

g) 安全防护（可选）：指功能提供者需执行的安全措施，常规的如数据正确性校验、缓存越界检查等。

h) 约束（可选）：描述此功能的使用条件，无约束时可省略。

7.1.2 Information ID 模型定义

每一个 Information ID 代表一种信息，与 Module ID 结合可组成不同模块上的对应信息。

a) Information ID：符合Information ID格式定义、无重复的ID，以十六进制格式书写。

b) 信息描述：对信息含义进行清晰无歧义的说明。

c) 信息格式：信息的详细格式说明，要求格式自对齐，不依赖运行环境。

Notify ID Information 的数据格式有两种定义途径，一种是遵循附录 A 的独立对象描述，另一种是简单的数据结构定义。

对象描述格式的优点是自带格式，对字段进行扩展或者删除均不影响其他字段，缺点是占用更多空间（Header），且需要依赖解析代码。

简单数据结构的好处是数据处理简单，占用空间小，缺点是扩展或删除可能会影响到其他字段。

对象描述格式，示例如下：

```
ubios_ob_v1("example_info ", total_size) {
    head@u32 = value
    tail@u64 = value
}
```

简单数据结构，示例如下，见表 4。

表 4 简单数据结构示例表

字段	偏移量（字节）	大小（字节）	描述
head	0	4	/
reserved	4	4	保留
tail	8	8	/

注：如果采用简单数据结构，数据格式应自对齐如果采用附录 A，应由相应的操作保证对齐访问。

7.1.3 Module ID 模型定义

每一个 Module ID 代表一个操作对象或信息产生源。

Module ID 共 16bits，包含 Main Module ID 和 Sub Module ID 两部分，见表 5。

表 5 Module ID 定义

bits [15:4]	bits [3:0]
Main Module ID	Sub Module ID

各字段说明如下：

a) Main Module ID：12bits，表示模块分类，最大支持4096种模块；

b) Sub Module ID：4bits，表示模块的子模块，值为0表示模块整体，从1开始表示子模块，每个模块最大支持15个子模块，子模块的设置可以为每个main module ID预留一段空间供未来扩展或用户自定义。

注：

a) 每次向本文件申请Module ID时，应先申请Main Module ID。Sub Module ID支持预留，也支持按需申请。

b) 按照Sub Module ID定义，最大支持15个，如果超过15个，需要申请多个Main Module ID。

7.2 Call ID 定义

Call ID 是 Message ID 中的一种，定义见表 6。

表 6 Call ID 定义

bits [31:28]	bits [27:12]	bits [11:0]
0xC	Module ID	Function ID

7.3 Notify ID 定义

Notify ID 是 Message ID 中的一种，定义见表 7。

表 7 Notify ID 定义

bits [31:28]	bits [27:12]	bits [11:0]
0x4	Module ID	Information ID

7.4 User ID 格式

UVB 支持多组件并发使用，存在资源抢占操作，当使用虚拟总线进行消息传递时，需要通过 User ID 来区分资源占用者，避免使用冲突。

多个组件可能支持相同的 Call ID Service，通过 User ID 区分响应者，防止冲突。

多个组件可以采用相同的 Notify ID Information 来报告信息，信息的接收者可能是唯一也可能不唯一，通过 User ID 来区分，表达信息上报的意图，防止冲突，例如：

a) BIOS 上报的某个 Notify ID Information 有以下三种情况：上报给 OS，上报给 BMC，以及同时上报给 OS 和 BMC，通过设置接收者 User ID 区分；
b) 一个异步的 Call ID Service 需要响应者通过 Notify ID Information 来报告执行结果，当多个组件同时通过虚拟总线调用该 Call ID Service 时，响应者也需要通过 User ID 来区分响应的接收者，避免错误。

User ID 的格式由两部分组成，共 32bits，见表 8。

表 8 User ID定义

bits [31:24]	bits [23:0]
Type	Index

各字段详细描述如下：

a) Type：8bits
0xFF - 表示不指定 User；
0x30 - Trusted OS，指运行在安全世界的 OS；
0x20 - Rich OS，指运行在普通世界的 OS；
0x11 - 本 UBPU 内的 Entity
0x10 - 其他 UBPU，包括其他 UBPU 内的 Entity
0x0B - BMC；
0x01 - BIOS；
0x00 - 不使用；
其他 - 保留；
b) Index：24bits
各类型内部序号，0 表示该类型整体，其他由各类型自行定义，可以类型内共用不做区分，也可以按需进行区分。当进行区分时，不要求同一个 User 在不同环境下 Index 相同，只要保证在本系统内、本类型内唯一即可。
注：User ID 的值 0 保留不用，不能再作为 User ID 使用。

8 功能接口定义

8.1 概述

UBIOS功能接口层是功能的集合。本层的同类功能均采用统一的接口格式，分为Call ID Service接口和Notify ID Information接口两种。

8.2 Call ID Service 接口定义

Call ID Service 是接收消息，根据消息内容执行功能或服务，并返回结果，是功能的提供者，数据包含输入和输出双向的。

Call ID Service 采用函数化实现方式，即编程上跨组件 Call ID Service 功能调用与本组件内函数调用类似，通过 UBIOS 信息交互层屏蔽具体数据通路上的差异。

函数调用行为包含 2 种：设置数据（或参数）和获取数据。 Call ID Service 接口应包括单向传递的参数、用于获取数据的缓存，以及函数的执行结果，函数原型如下：

```
/* Parameters:
```

```
   input        - option, parameter struct pointer, exact parameters according
                  Function definition, input NULL if no input parameter
   inputSize    - input buffer or paramters size
   output       - option, used to store returned data, input NULL if no output data
   pOutputSize  - when in input, it is output data buffer size
                - when in output, it output data size

   Returned Status:
   U_Status     - 0 means success, other value means failed
   */
   U_Status CisFunction(const void *input, unsigned int inputSize, void *output,
unsigned int *pOutputSize)
```

参数说明如下：

input和inputSize：输入参数，由具体功能函数的定义决定，通常是一个结构体；

注：input里不能包含指向另一个区域的指针，因为跨组件调用时，彼此对地址的访问权限可能存在差异，未必能直接访问，如果需要接收方函数去感知通信双方、感知通信方式、感知地址属性而区别对待，则会加重函数实现的复杂度，增加负担。

output：用于接收获取的数据，该output data buffer由UBIOS信息交互层根据实际情况进行处理，使两边UBIOS功能接口层不感知系统差异。

pOutputSize：作为输入时，用于存储output data buffer的大小，作为输出时，它存储output data的大小。

注：

a) input和output都是可选的，视具体的功能定义而定。

b) input信息为只读，不能通过修改input传递返回结果，output为只写，不能通过output传递入参。

c) pOutputSize所指向的值在输出时表示实际数据大小，如果值大于输入时，则返回的U_Status必须表示空间不足。

Call ID Service 接口也是 UBIOS 功能接口层与 UBIOS 信息交互层之间的接口之一，UBIOS 信息交互层根据 Call ID 定位具体的功能接口函数，并以符合上述函数原型传入参数进行功能调用。

8.3 Notify ID Information 接口

Notify ID Information 是单向信息传递，响应只包含应答，表明已收到，不返回数据。

作为提供者只产生数据，并将数据发送给目标接收者，数据格式定义在对应的 Information ID 中描述。

作为接收者，应根据需要进行 Notify ID Information 处理（Callback），函数原型如下：

```
   /* Parameters:
   Input      - Notify ID Information data pointer, exact format according Information
                definition
   inputSize  - input data size

   Returned Status:
   U_Status   - 0 means success, other value means failed
```

```
Note: the input = NULL is valid, means as a signal.
*/
U_Status NiiCallback(const void *input, unsigned int inputSize)
```

注：各组件可以自行注册各种 notify ID information 的回调，注册功能由信息交互层提供。从框架层面，input 可以为空，意味着需要各回调函数（根据对应的 Notify ID 决定）自行判断是否允许 input 为空。

9 信息交互层

9.1 信息交互层接口

UBIOS 信息交互层接口包含面向使用者、面向其他信息交互层、面向功能接口层，见图 9。

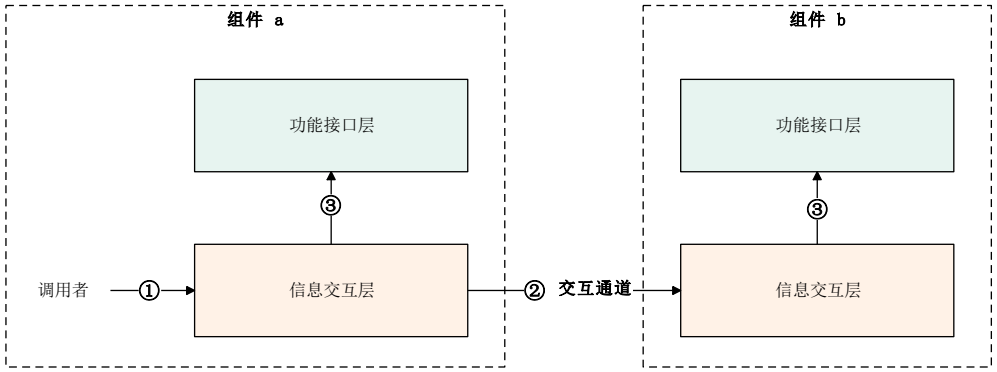

图 9 信息交互层接口

信息交互层接口步骤说明：

① 调用者通过过信息交互层的接口发起Call ID Service调用；

② 组件a通过组件间信息交互层通接口向组件b传输信息；

③ 信息交互层查询并调用功能接口层的功能；

调用流程示例详细说明见章节6.5 。

面向使用者又可根据目的差异分为面向Call ID Service调用的接口和面向Notify ID Information发送的接口。

9.1.1 调用 Call ID Service 函数接口定义

面向 Call ID Service 调用的接口，即使用者可以通过信息交互层的这一接口发起 Call ID Service 调用（图 14 ①所示），接口定义如下：

```
/*
Parameters:
callId      - the ID of this Call ID Service
receiverId  - the expectant responser user ID of this Call ID Service call
other parameters    - see Call ID Service interface definition

Returned Status:
U_Status    - 0 means success, other value means failed
```

```
*/
U_Status CisCall(unsigned int callId,
           unsigned int receiverId,
           const void *input, unsigned int inputSize,
           void *output, unsigned int *pOutputSize)
```

参数说明如下：

callId：本次需要调用的Call ID Service。

receiverId：响应者User ID，即指示本次调用希望被谁响应，可以不指定。

input和inputSize：即Call ID Service功能接口入参。

output和pOutputSize：即Call ID Service功能接口出参。

注：调用侧无需关注跨组件地址感知差异，除了不支持指针嵌套，调用时对 input 和 output 无额外要求，由信息交互层按需处理。

9.1.2 发送 Notify ID Information 函数接口定义

面向 Notify ID Information 发送的接口，即使用者可以通过信息交互层的这一接口发送 Notify ID Information，接口定义如下：

```
/*
Parameters:
notifyId    - the ID of this Notify ID Information
receiverId  - the expectant receiver user ID of this information
info        - information pointer, exact information data according Information
              definition, NULL is valid.
infoSize    - information size, must > 0
needAck     - indicate if this NII needs a ack from receiver, true means yes, false
              means no

Returned Status:
U_Status    - 0 means send success, other value means failed
*/
U_Status NiiSend(unsigned int notifyId,
           unsigned int receiverId,
           const void *info, unsigned int infoSize,
           bool needAck)
```

参数说明如下：

notifyId：本Notify ID Information对应的ID。

receiverId：接收者User ID，即指示本次消息希望被谁接收，必须指定，信息交互层可基于接收者ID决定采用什么交互通道进行消息传递。

info和infoSize：指向消息数据的指针及消息数据长度。

9.1.3 信息交互层间接口

信息交互层之间的接口指的是组件间信息交互层进行通信的接口。

当信息交互层向另一个组件发送消息时，会根据实际所需传递的消息以及目的地选择所用的交互通道，再由交互通道决定传递的数据格式。

当信息交互层接收来自另一个组件的消息时，会检测所有可能收到消息的交互通道，识别是否存在发送给自己的消息，再根据消息 ID 交由功能接口层处理。

信息交互层的接口定义应符合“交互通道定义”章节。

注：每个组件的 User ID 由信息交互层维护管理，功能接口层通常可不关注，除非存在特殊需求。

9.1.4 功能接口层回调接口

功能接口层回调接口指信息交互层收到调用请求后，查询并调用功能接口层的功能，遵循功能接口层的接口定义。

注：信息交互层也可以支持本地调用，可根据 receiverId 确定是否从本地查询响应函数。

9.2 交互通道选择

9.2.1 面向 Call ID Service 调用

在本文件中定义了 Call ID Service Table 和 Receiver User ID 来确定彼此信息交互的通道。

Call ID Service Table 描述了每一个 Call ID Service 的信息交互通道，同时也可以用于快速过滤不支持的 Call ID Service 调用。当一个组件通过 UBIOS 信息交互层发起一次 Call ID Service 调用时，信息交互层首先查询 Call ID Service Table 中对该 Call ID 的描述，如果查不到或显示不支持，则立即返回，否则采用表中所描述的方式进行信息交互。

User ID 则用来区分 Call ID Service Table，在计算系统中，每个组件既是服务的使用者也是服务的提供者，一个组件可以调用多个组件的 Call ID Service，每个组件支持哪些 Call ID Service 以及对应的 Call ID Service 采用何种交互方式，只有该组件自己知道，因此：

1. Call ID Service Table表示一个组件对外提供的服务信息，由该组件对外提供；
2. 一个组件可以获得来自多个组件的不同Call ID Service Table，通过不同的User ID来进行区分。

注：Call ID Service Table 详细内容见 UBIOS 信息表章节。

通过 UBIOS 信息交互层发送消息示意见图 10。

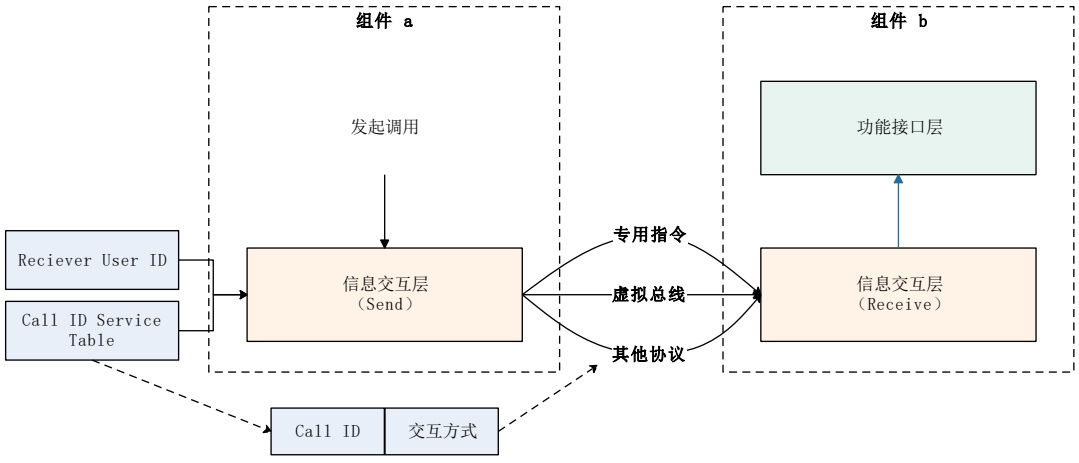

图 10 信息交互层发送消息示意图

注：消息返回时按原路返回，无需再次查询 Call ID Service Table。

9.2.2 面向 Notify ID Information 发送

当前 Notify ID Information 只支持由 BIOS 给 OS 发送、由 BIOS 给 BMC 发送、由外设给主机发送 3 种场景。

a) 对于 BIOS 给 OS 发送 Notify ID Information 场景，通道选择方式由 BIOS 实现决定。
b) 对于 BIOS 给 BMC 发送 Notify ID Information 场景，通道由 BIOS 与 BMC 之间的物理通道决定。
c) 对于外设给主机发送Notify ID Information场景，通道由外设与主机之间的物理通道决定。

9.3 参数地址处理

9.3.1 概述

在进行跨组件功能调用或信息传递时，不同组件的地址译码可能会存在差异，为了使功能开发者无需过多关注这些差异，信息交互层需要对参数中的地址进行妥善处理。

注：不支持指针嵌套，即不支持指针指向的数据中又包含指针指向另一段数据。

从地址处理维度看，跨组件信息交互可以分为 2 种方式：涉及地址处理的有共享交互和不涉及地址处理的无共享交互。

a) 有共享交互：表示两个组件间存在共享空间，在进行消息传递时传递的是指向消息内容的地址或指针，消息内存存储在共享空间内，无需进行拷贝搬移。
b) 无共享交互：表示两个组件间进行消息传递时直接传递消息内容，不依赖彼此共享空间。

采用何种交互方式一方面依赖底层硬件，有的硬件通路不支持有共享交互，另一方面也跟消息传递的效率、安全性有关，本章节描述有共享交互方式下的参数地址处理。

在有共享交互方式下，UBIOS规范采用2种参数地址处理方式：透传模式和转存模式。

9.3.2 透传模式

透传模式下，信息交互层不对参数地址做任何处理，参数直接传递给接收者。见图 11。

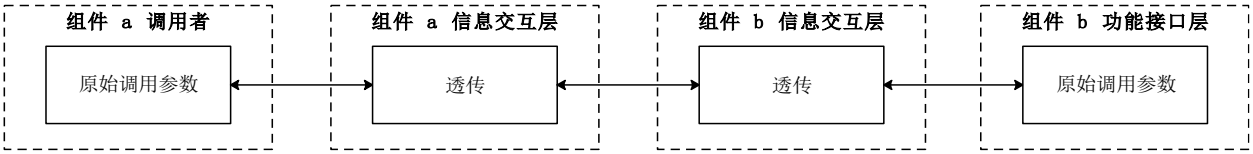

图 11 透传模式示意图

这种方式效率最高，也最简单，但仅适用于交互双方地址映射相同的情况。示意见图 12。

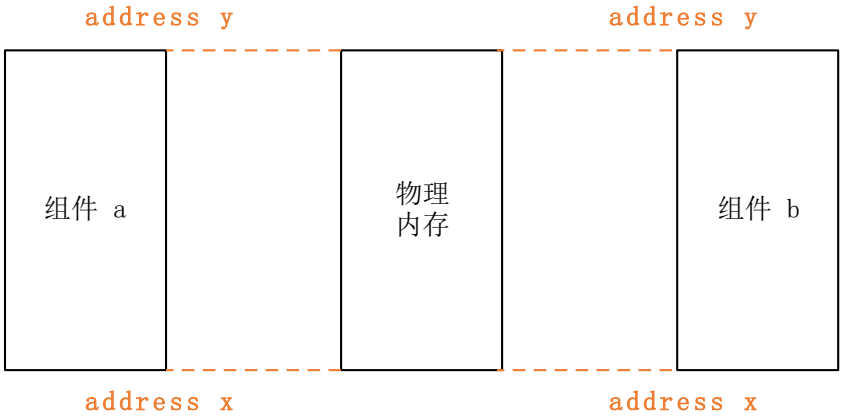

图 12 透传模式下内存地址映射示意图

交互双方对物理内存的地址映射相同，一个组件的参数（指针）在另一个组件可以得到完全相同的解析，所以可以采用透传模式。

9.3.3 转存模式

转存模式指信息交互层对参数内容进行搬移，并更新参数地址，然后才传递给另一个组件，当有消息从另一个组件返回时，再对参数地址进行还原，使功能接口层对地址转存无感，帮助功能接口层屏蔽地址差异。

同一块共享内存在 2 个组件中映射的地址可能不同，地址映射关系见图 13。

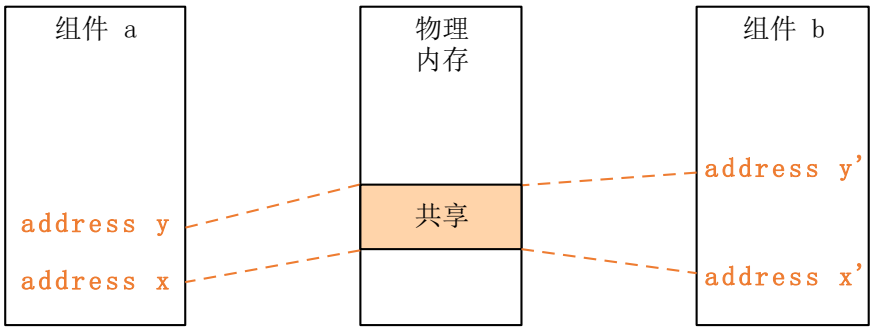

图 13 转存模式下内存地址映射示意图

注：上图中地址映射差异不仅仅指共享部分，还包括其他非共享部分的映射或访问权限差异，只要不满足完全相同，则不能采用透传模式。

这种存在地址映射差异的情况下只能采用转存模式，执行流程见图 14。

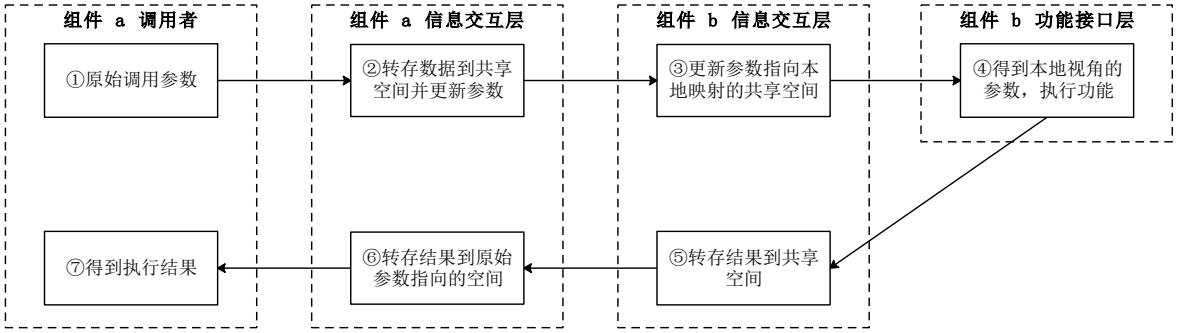

图 14 转存模式下执行流程示意图

① 组件a的调用者像调用本地函数一样准备参数；
② 组件a信息交互层掌握共享空间地址，且能访问组件a本地地址，将数据搬移到共享空间，并更新参数指向共享空间的物理地址（而非组件a本地映射的地址）；
③ 组件b信息交互层掌握共享空间地址，根据地址映射差异将来自组件a的共享空间物理地址更新为组件b可直接访问的地址；

注：组件之间交互需采用主系统视角的物理地址。

④ 组件b功能接口直接使用来自组件b信息交互层的参数，与被本地程序调用一样；
⑤ 组件b功能接口层已按需将返回数据写入了共享空间，组件b信息交互层将参数地址转回到共享空间的物理地址传递给组件a；
⑥ 组件a信息交互层从共享空间取出数据转存到原始参数指定的地址；
⑦ 组件a调用者从原始参数指定的地址获取返回数据，如从调用本地函数返回一样。

这种方式可以屏蔽组件间地址译码差异，同时提供一定的安全性（限定数据传递所使用的地址范围），并为使调用者和功能接口层屏蔽组件地址映射差异。

注：如果交互双方存在专用的共享缓存，则在交互过程中可以免去地址转换的操作，发送侧和接收侧都可以基于自己的地址译码操作专用的共享缓存，这是优选方案，如 UVB 中的专用 buffer。

9.3.4 模式选择

参数地址处理模式选择存在多种因素，包括物理环境、安全需求、效率等，如何选择由 BIOS 开发者决定，但如何表达选择结果则需要在规范中进行约定，内容在相应的交互通道信息表中描述。

内存虚拟总线交互方式需进行模式选择（在UVB信息表中描述），芯片ISA交互方式默认采用透传模式，其他协议当前仅支持无共享方式交互。

9.4 Notify Interrupt

在通过 UVB 进行消息通信时，除了轮询之外，还支持借助中断来实现通知功能，在本文件中称为 Notify Interrupt。在同一个计算系统中，Notify Interrupt 可以根据接受对象不同设置多个，记为 Notify Interrupt 接受对象，无论包含多少，Notify Interrupt 接收对象只代表知会接受对象 UVB 上有新消息，本身不包含具体消息内容。

Notify Interrupt(OS)：具体在Notify Info信息表中描述。

Notify Interrupt(其他)：暂不包含在本文件之中，由开发者自定义。

10 交互通道定义

10.1 通过芯片 ISA call 的交互

10.1.1 概述

通过芯片ISA call的虚拟总线交互即通过芯片ISA call进行交互的一种虚拟总线实现方式，当前只支持Call ID Service，下面内容均为Call ID Service信息交互定义。

10.1.2 AArch64

基于 ARM 的 64 位指令集（AArch64）的虚拟总线交互应通过通用寄存器传递调用信息，并通过 SMC 指令实现调用，指令格式定义见表 9。

表 9 SMC 指令格式定义

指令	立即数	输入	输出
SMC	0	W0 ： CIS_CALL_SIGNATURE X1 ： 参数基地址（或指针）	W0 ： 返回状态

CIS_CALL_SIGNATURE：区别于SMC其他命令的特殊标识，标识此次 通过SMC调用的是Call ID Service，定义如下：

#define CIS_CALL_SIGNATURE 0xEF00F0F0

参数基地址（或指针）：指向Parameter Structure的地址（或指针），Parameter Structure结构定义如下:

```
// Parameter Structure
struct IsaCallParam {
    unsigned int messageId;
    unsigned int senderId;
    unsigned int receiverId;
    unsigned int reserved0;
    const void *input;
    unsigned int inputSize;
    unsigned int reserved1;
    void *output;
    unsigned int *pOutputSize;
};
```

注：当直接通过 ISA Call 传参时，Parameter Base Address (or Pointer)指向数据结构为 struct IsaCallParam 的临时内存；当通过内存虚拟总线传参，Parameter Base Address (or Pointer)将指向内存虚拟总线中的某一个窗口。

调用指令:

SMC 指令，指令参数为 0

调用入参：

W0：即 X0 的低 32bits，填 CIS_CALL_SIGNATURE。

X1：对应的入参的内存物理地址。

调用返回:

W0：执行返回状态，详见 Call ID 对应的 Function ID 定义。

注：其他返回信息记录在入参指定的地方，不通过通用寄存器传递。

10.2 通过内存虚拟总线交互

10.2.1 概述

UVB 仅能由 BIOS 各固件、OS kernel 及以上权限的软件访问。BIOS 作为外设、BMC 等外部组件向内存虚拟总线发送消息的代理。BIOS 可对来自外部组件的消息进行安全检验（如消息来源的身份认证等）后再转发。 OS kernel 及以上权限的软件应作为 OS 侧发往内存虚拟总线消息的代理，用于保障

进入总线的数据具备不低于 BIOS 和 OS Kernel 的权限，内存虚拟总线无需再进行安全等级检查，由数据使用者自行判断内存虚拟总线上消息的合法性。

内存虚拟总线的数量可按需设置，不受限制。存在更高安全要求的情况下，宜增设一条安全的内存虚拟总线，用于安全组件间的互相调用；对于某些支持硬件通道的特定组件，可单独设置一条内存虚拟总线；对于 Notify ID Information 上报场景，也可按消息紧急程度，设置多条内存虚拟总线来传递。

每条内存虚拟总线由 1 个或 1 个以上窗口（window）组成，窗口是进行一次消息传递的最小单元，窗口可以是离散分布的，也可以是连续的，通过软件将其组成一条连续的虚拟总线，见图 15。

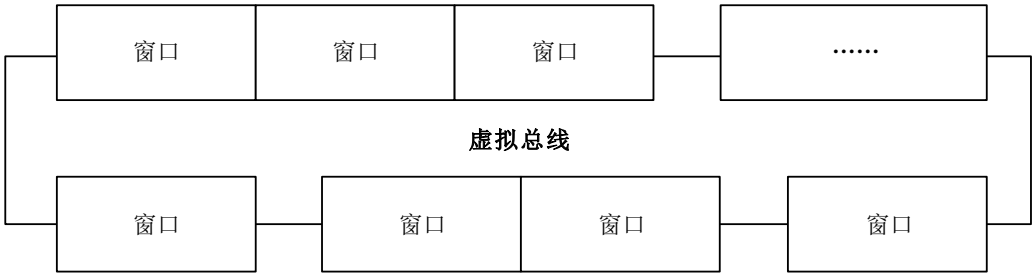

图 15 内存虚拟总线窗口

注：UVB窗口的安全属性决定了通信的安全属性。对于高安全要求的通信，在只有高安全等级才能访问的UVB窗口进行，安全属性的窗口在普通权限下无法访问。

10.2.2 虚拟总线窗口格式定义

每个内存虚拟总线窗口长度固定为 64 字节，其基本格式定义见表 10。

表 10 虚拟总线窗口基本格式定义

字段	偏移量(字节)	大小(字节)	说明
Version	0	1	窗口数据格式版本，当前为 1
Flag	1	1	功能标志: bits [7:2]: 保留位 bits [1:0]: 帧状态 0 : 只有一帧 1 : 有多帧，该帧不是最后一帧 2 : 该帧是最后一帧 3 : 确认帧，表示已收到，可以发送下一帧 **注：** 详见 8.2.4 分包传输。如果未使用特定缓冲区，则忽略该字段。
Reserved	2	2	保留字段

字段	偏移量(字节)	大小(字节)	说明
Message ID	4	4	用于区分消息的内容类型 Call ID: 表示这是一条 Call ID Service 消息 Notify ID: 表示这是一条 Notify ID Information 消息 Call ID 补码：表示这是一条 Call ID Service 响应 Notify ID 补码：表示这是一条 Notify ID Information ACK 响应 其他值: 未定义 **注：** a) 响应消息时，必须在写入该窗口的所有其他字段之后，最后写入该字段。 b) 当传输完成时（收到返回状态、收到确认或超时），发送方必须将此字段清除为 0。
Sender ID	8	4	发送该消息的发送者的 User ID，也可以作为该窗口的获取标志。 0：空闲，可分配用于发送消息 User ID：正在处理消息发送 其他值：未定义 **注 1：** 如果用它来占用这个窗口： a) 发送方在向该字段写入值之前必须检查该字段并确保其为 0。 b) 释放窗口时必须最后清除该字段。 **注 2：** 发送消息时，必须在写入本窗口的所有其他字段之后，最后写入该字段。
Receiver ID	12	4	该数据接收方的 User ID。
Input Data Address	16	8	输入数据区起始地址，数据格式由所传递的信息决定，在各接口描述中定义。如果没有专属 buffer，必须是系统物理地址，32 位系统时[32, 63]位保留；如果有专属 buffer，忽略非 0 值。 0：没有输入数据 其他值：表示输入数据基地址。
Input Data Size	24	4	输入数据的总长度，如果没有输入数据（即 input data address = 0）则忽略该字段。
Input Data Checksum	28	4	计算全部输入数据的 4 字节之和，然后在此处填写。 如果没有输入数据，则忽略此操作。 **注：**当输入数据大小与 4 字节不对齐时用 0 填充。
Output Data Address	32	8	输出数据区起始地址，数据格式由所传递的信息决定，在各接口描述中定义。如果没有专属 buffer，必须是系统物理地址，32 位系统中[32, 63]位被保留；如果有专属 buffer，忽略非 0 值。当 Message ID 为 Notify ID 时，忽略此项。 0：无输出数据 其他值：表示输出数据基地址。 **注：**当不需要取回数据或没有输出数据时，可以选择此选项。
Output Data Size	40	4	当 Message ID 为 Call ID 时，表示输出数据空间的大小；当 Message ID 为 Call ID Service 的响应时，表示输出数据的确切大小；当 Message ID 为 Notify ID 时，忽略此值。 如果没有输出数据则忽略此字段。

字段	偏移量(字节)	大小(字节)	说明
			注：值 0xFFFFFFFF 表示该字段无效。
Output Data Checksum	44	4	计算全部输出数据的 4 字节之和，然后在此处填写。 当 Message ID 为 Notify ID 时，忽略此项。 如果没有输出数据则忽略此字段。 **注：**当输出数据大小与 4 字节不对齐时用 0 填充。
Returned Status	48	4	用于返回执行状态。如果不是响应消息，请忽略此消息。 用于 Call ID Service 调用时忽略此字段。
Reserved	52	8	保留字段
Forwarder	60	4	消息转发者的 User ID

各个UVB窗口相互独立，每个窗口都可用于传递Call ID Service或Notify ID Information，Call ID和Notify ID的设计可以保证彼此不会出现混淆。

10.2.3 虚拟总线窗口的占用

内存虚拟总线由多组件共享，每个组件均可对其进行读写操作。组件进行写操作前，应对窗口进行互斥检查，通过 User ID 确认自身是否占用该窗口。窗口的写操作，应遵守查询、占用、使用、释放的步骤，读操作不受此约束。窗口的共享属性（读写时间差）可造成虚假占用问题，应通过以下方法规避：

纯软件窗口占用方法：

当使用者检测到空闲窗口时，写入自己的 ID，并延时一段时间，再回读确认是自己的 ID 表示占用成功，否则占用失败。见图 16。

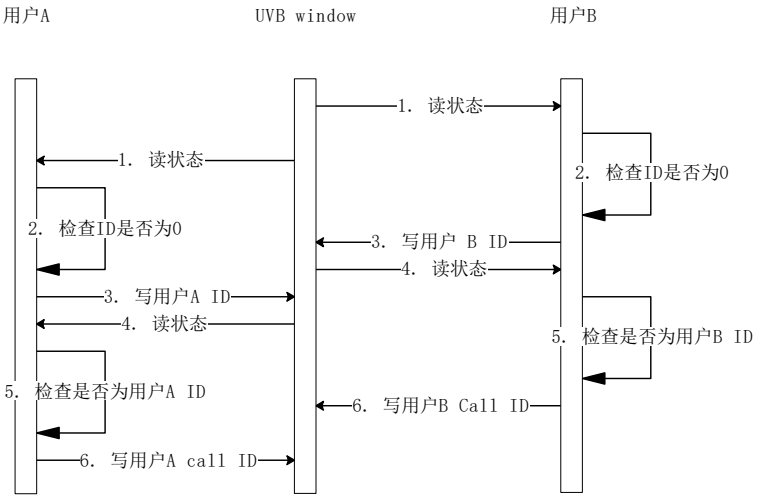

图 16 纯软件窗口虚假占用场景

上图步骤 3、4 之间的延时时间至少要大于步骤 1 到 3 所需的时间，建议 1 之前加锁，5 之后释放锁，防止被打断导致时间超预期，步骤 3、4 之间增加延时，具体延时要求在 UVB 信息表中描述。

硬件辅助窗口占用方法：

由硬件为每个窗口提供一个 32bits 位宽的占用寄存器，该寄存器满足值为 0 时可写入任意值，值不为 0 时只能写 0，使用者写入自己的 ID 后可无需延时，直接回读并判断是否写成功。

10.2.4 虚拟总线窗口的释放

内存虚拟总线窗口是共享的，使用者完成使用后应将窗口各字段（表示占用的字段外）设置为缺省值以免数据残留，再将表示占用的字段写为 0 释放该窗口。

内存虚拟总线窗口采用谁发起谁释放的方式。每一次发起如得到响应，应在完成响应消息处理后释放窗口；如未获得响应，则应在达到超时时间后释放窗口。要求如下：

a) 默认情况下下，内存虚拟总线窗口发送消息的超时时间不应超过1s，对于消息处理时间大于1s的服务，应当采用异步方式，先返回响应，再处理，处理后再返回结果。

b) 根据业务实际情况，BIOS虚拟总线的消息接收可采用轮询模式，也可采用中断模式，或者两者结合。轮询模式是按总线设置遍历每个窗口；中断模式可以是汇聚成一个中断，如Notify Interrupt，触发后启动轮询，也可以是每个窗口设置自己的中断。

10.2.5 分包传输

当实际传输的数据超出对应窗口的专属缓存大小时，组件应将数据分为多帧在 UVB 上进行传输。由于传输过程中窗口是独占的，可以保证顺序，所以窗口 flag 中只需要表示帧的状态。

根据所传输数据（input 和 output）长度是否超过专属缓存（buffer）来判断是否需要分帧，分包传输分为以下 5 种场景：

a) 场景1. 不需要分帧：input和output数据总和小于或等于buffer大小

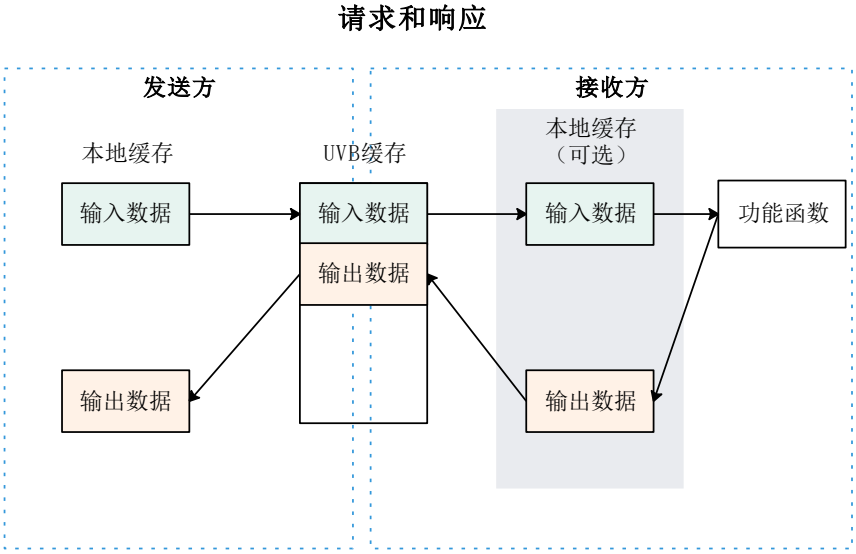

图 17 不需要分帧场景

如图 17 所示，在不分帧的情况下，input 和 output 复用同一个 buffer 进行传输，此时 receiver 可选择将数据转存后再交给功能（function），也可不转存，直接交给功能。

此场景下，发送请求和返回相应时 flag.frame 都是**不分帧（0）**。

b) 场景 2. input 需要分帧， output 不需要分帧：input 数据大小超过 buffer 大小，output 数据大小未超过 buffer 大小

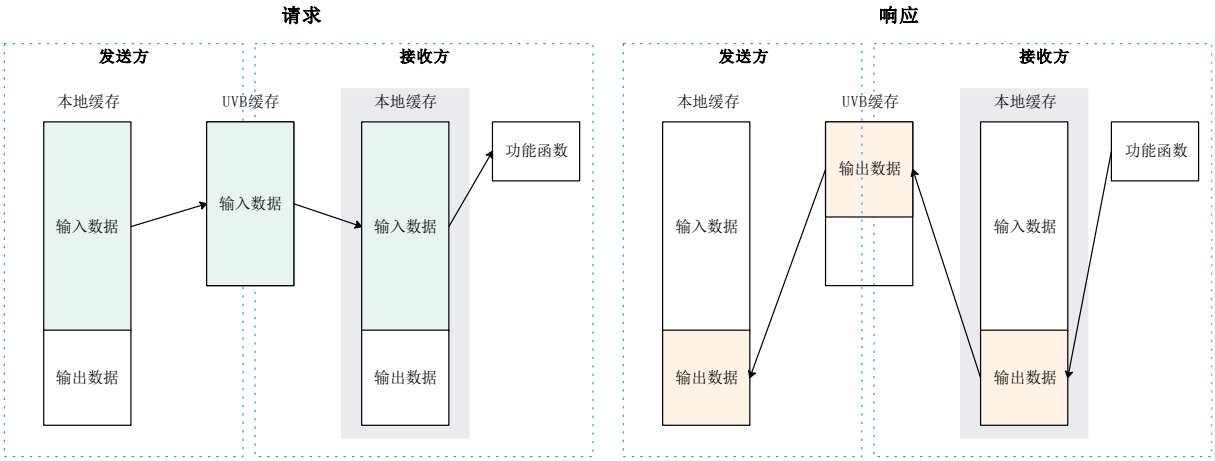

图 18　input 需要分帧，output 无需分帧场景

如图 18 所示，input 数据超过 buffer 大小，需要分帧传输，receiver 必须分配缓存来多次接收；当返回 output 数据时，它的大小小于 buffer 大小，所以不需要分帧。

此场景下，发送请求时 flag.frame 为“分帧未结束（1）”，直到发送 input 数据的最后一帧时改为“分帧已结束（2）”；响应请求时，每完成一帧数据（非结束帧）接收将 flag.frame 为 “已读数据请继续（3）”，当接收完结束帧时，开始执行，并返回执行结果，即 output，返回时 flag.frame 为“分帧已结束（2）”。即使 output 不需要分帧，receiver 在申请缓存时同时为 output 申请缓存，并且在返回时，output 会直接放在 UVB buffer 的起始，这一点与场景 1 不同。

c)　场景 3. input 不需要分帧， output 需要分帧：input 数据大小未超过 buffer 大小，output 数据大小超过 buffer 大小

过程与场景 2 类似，差别是 input 数据只需要一帧就可以发送完成，此时 flag.frame 为“分帧已结束（2）”，在前面没有 “分帧未结束（1）” 状态。在返回 output 数据时，与场景 2 发送 input 数据完全一致。

d)　场景 4. input 和 output 都需要分帧：input 和 output 数据大小均超过 buffer 大小

即场景 2 的 input 数据传输和场景 3 的 output 数据传输组合。

e)　场景 5. input 和 output 都不需要分帧，但加起来需要分帧：input 和 output 数据大小总和超过 buffer 大小

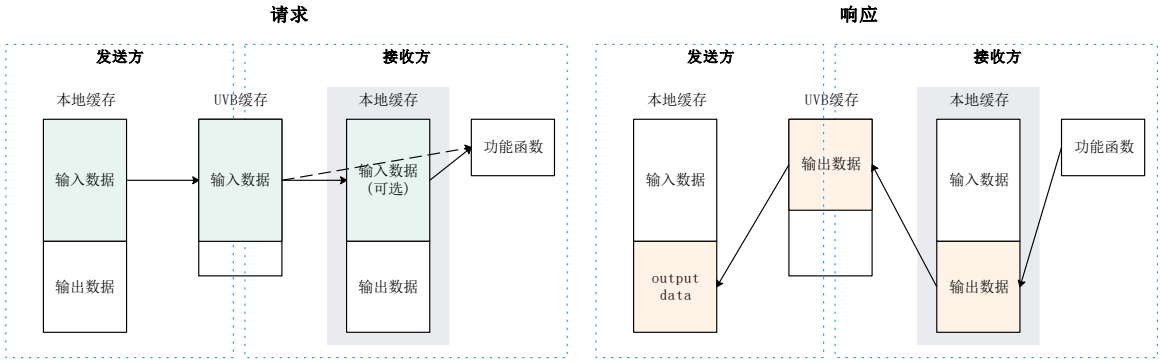

图 19　input 和 output 都不需要分帧但加起来需要分帧场景

如图 19 所示，场景 5 与场景 1 的关键差别是，每次传输都只需要一帧就可以传完，但仍然需要 receiver 分配缓存来暂存 output 数据。

此场景下，发送请求时 flag.frame 为“分帧已结束（2）”，和响应请求时 flag.frame 为“分帧已结束（2）”。

综上 5 种场景，除了场景 1，其他场景都需要分配缓存，每一帧的 input 数据和 output 数据都在 UVB buffer 的起始。如果发送时采用分帧，那么成功返回数据时也必须时分帧，即使返回数据只有 1 帧。基于此，sender 一旦采用分帧发送，等待响应数据时，必须先等到执行响应才会处理返回数据，必须等到帧状态为“分帧未结束（1）”或“分帧已结束（2）”才会读取数据，receiver 会利用这一点先将帧状态设为其他（如“已读数据请继续（3）”），再返回执行状态，再发送数据。

10.3 通过其他协议的交互

10.3.1 消息承载机制

其他协议泛指芯片 ISA 交互、内存虚拟总线之外，已具备格式定义的通信协议， 可使用这些协议报文中的数据段传递 UVB 消息。

如果协议的每一个数据报文已经有严格的定义，不支持对数据进行扩展，则需要增加一种新的数据报文类型来支持本文件的应用。

注：针对特定协议需要新增或修改时，修改的具体内容将在对应协议进行描述。

采用此方式进行消息传递时涉及 3 部分内容：

a) 标识：按照相应协议要求，定义一个标识来标识此次发送的消息是传递UBIOS规范的消息，该标识可以是在相应协议原有标识上扩展（即增加一种表示UBIOS的标识），或在协议的Data或payload字段中传递。

b) 发送数据：按照下述各种数据格式作为协议的Data（或payload）字段填入报文。

c) 接收数据：按照下述各种数据格式解析协议收到的Data（或payload）字段内容。

注：

a) 通过其他协议传递数据，由对应协议保证数据可靠，不严格要求额外增加校验信息，各功能接口是否设置校验可自选。

b) 协议需要能够提供Data（或payload）字段的长度。

在采用此类协议进行实际信息交互中，不同的系统对数据的处理存在多种可能，有的依赖报文传递所有数据，有的支持通过DMA获取数据或发送数据，报文中只需传递数据地址，有的支持地址空间共享，可以用相同的地址表达，有的不支持地址空间共享，需要进行地址翻译，为了避免因为需要支持多种模式而定义过多冗余字段，选择通过type来进行区分，定义多种type的数据结构，按需选择使用。当前只涉及通过报文直接传输数据场景，本文件只对该type进行定义。

10.3.2 Type1：通过报文直接传输数据

此类型适用于通过报文传递数据的场景，通过该方式进行消息传递时，参数数据和执行状态都需要通过报文传递，报文中不含有指向另一段数据的地址或指针，格式定义见表 11。

表 11 通过报文直接传输数据格式定义

字段	偏移量（字节）	大小（字节）	说明
Type&Version	0	1	bits [3:0]:表示此 data 数据结构类型，值为 1 bits [7:4]:表示报文格式版本号，值为 1
Reserved	1	1	表示此 data 数据结构类型，此类型值为 1
Index	2	2	帧序号，从 0 开始，分帧时有效，Data Total Size 大

			于单次可传递数据时会进行分帧
Message ID	4	4	用于区分消息的内容类型： • Call ID: 表示这是一条Call ID Service消息 • Notify ID: 表示这是一条Notify ID Information消息 • Response ID: 表示这是一条响应消息
Sender ID	8	4	发送该消息的发送者的 User ID，见 User ID 定义
Receiver ID	12	4	接收该消息的接收者的 User ID，见 User ID 定义
Output Size/Returned Status	16	4	• 当Message ID是Call ID时，表示Output Size，如果无效（即pOutputSize = NULL），用0xFFFFFFFF表示； • 当Message ID是Response ID（即Call ID Service的返回消息）时，表示执行结果，相当于函数返回值，实际Output Size由Data Total Size字段表示； • 其他消息忽略。
Data Total Size	20	4	表示数据总长度，数据可能分成多帧发送；可以通过已接收数据长度和与此字段比较来判断是否已接收完，如果没有返回数据则该字段为 0。 • 当 Message ID 为 Call ID 时，此字段表示 Input Size，即输入数据的总长度； • 当 Message ID 为 Call ID Service 的响应时，此字段表示 Output Size，即实际输出数据的总长度，应该小于等于 Output Buffer Size，如果无输出或 Output Buffer Size 无效，此处应为 0； • 当 Message ID 为 Notify ID 时，此字段表示 Input Size，即传输的 Information 数据的总长度。
Data	24	-	实际传输的数据，当Data Total Size表示Input时，为Input数据，当Data Total Size表示Output时，为Output数据。

11 UBIOS 信息表

11.1 概述

UBIOS 信息表为本文件中涉及的各种信息表的统称。用户可自定义扩展描述，语法请参考附录 A。

11.2 信息表基本规则

11.2.1 信息表数据传递方式

依据信息传递场景不同，信息表数据传递方式可分为共享内存和协议报文 2 种，要求如下：

a) 共享内存

共享内存指采用内存作为共享空间来传递 UBIOS 信息表，通常应用于 BIOS 与 OS 之间，根据实际使用需求，共享内存包含临时性共享内存和持续性共享内存。

1) 临时性共享内存：用于BIOS向OS主动上报信息表；

2) 持续性共享内存：用于BIOS向OS主动上报Notify ID Information。

b) 协议报文

协议报文属于非共享空间，指数据通过协议报文直接传输，通常应用于BIOS与BMC间通信。

11.2.2 信息表传递方式

依据信息传递的发起方式，信息表传递方式可分为主动上报和被动响应两种：

主动上报是指信息提供者在没有得到请求的情况下，主动向接收者上报信息。主动上报时可以上报一个表，也可以上报多个表。

被动响应是指信息提供者等待信息接收者发出明确指令。接收者主动调用Call ID Service获取UBIOS信息表，然后信息提供者才上报。所有信息表由Call ID Service提供者维护，信息表作为Call ID Service的返回结果输出，上报的信息表范围由调用参数决定。

上述两种方式不互斥，某个信息表可以支持其中一种，也可同时支持两种，Call ID Service通常只用于获取单个信息表。

11.3 系统信息表

11.3.1 概述

UBIOS 系统信息表指支撑框架功能正常运行所需的信息，通常是系统必备的，包括 root table、memory map、UVB 等，由一个索引总表和多个信息子表组成，每个信息子表都有自己的表头和内容，可独立使用，在通过 Call ID Service 获取指定信息子表时可单独传递。索引总表是传递完整 UBIOS 系统信息表的入口，通过总表可检索到各个信息子表。

UBIOS 系统信息表格式的公共规则：

a) UBIOS系统信息表采用附录A定义的格式进行描述，同时兼容拥有相同子表表头的其他格式信息表，索引总表即对象描述根，每个信息子表都是一个独立的对象描述文件；

b) 每个表都有一个独立、不重复的名字作为标识。

c) 每个表的起始地址8字节对齐。

11.3.2 UBIOS 信息索引表（总表）

UBIOS 信息索引总表是所有 UBIOS 信息表的根，即附录 A 所定义的对象描述根（Object description root）。

UBIOS 信息索引表描述如下：

```
ubios_od_root_v1("root_table", max_od_number) {
    object description
}
```

注：object description 仅表示此处包含各种对象描述，没有特指，只是示意表达。

Root Table对象描述根(od root)结构见表 12。

表 12 Root Table 根表结构

字段	偏移量（字节）	大小（字节）	说明

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“root_table”
Total Size	16	4	整个表的总大小（包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 **注：** a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。
Count	32	2	OD Files 数量
Reserved	34	6	保留字段
OD Files[Count]	40	8*Count	OD 文件的系统物理地址

注：

a) 索引总表仅作为表的索引，不包含其他信息，其他信息都应当在对应的对象描述中体现；

b) 索引总表不要求体现每一个对象描述文件，如部分对象描述可通过Call ID Service获得。

c) OD Files[Count]中有效地址可不连续存放，即可存在空隙，值为0表示空隙，查询时应当遍历。

d) Count指包含空隙在内的索引总表最大可索引的对象描述文件数量。

e) 针对拥有相同子表表头的其他格式信息表，在同一个root table中，不同类型的子表可共存。

11.3.3 内存映射

Memory Map 用于描述内存地址空间信息，Memory Map 的描述是用户视角，即报告者（Reporter）告知接收者（Receiver）有哪些内存，分别是什么类型，哪些类型可用、哪些类型不可用等。

Memory Map 所描述的所有地址都是物理地址。

Memory Map 仅支持临时共享内存空间方式传递，必须主动上报，是否支持通过 Call ID Service 获取可选。

在某些架构下，在获取 Memory Map 完成内存基础配置之前不支持非对齐访问，因此，Memory Map 信息以及获取 Memory Map 的过程需要保证按 8 字节对齐设计。

Memory Map 对象描述：

```
ubios_od_v1("memory_map", total_size) {
    memory_map@raw = {
        value
    }
}
```

注：value 仅表示此处填值，没有特指，只是示意表达。

Memory Map对象描述头(Object Description Header)结构见表 13。

表 13 Memory Map 描述头结构

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“memory_map”
Total Size	16	4	整个表的总大小（包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 注： a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

Memory Map对象描述成员列表见表 14。

表 14 Memory Map 队形成员描述表

名称	类型	是否必选	说明
memory_map	raw	是	每一个内存区域的信息

注：为了保证 Memory Map 数据地址 8 字节对齐，可以直接使用，避免非对齐访问问题，需保证：

a) memory_map字段必须放在对象memory_map的第一个位置，如果存在其他属性需放在后面（通常情况不会放其他内容）；

b) memory_map字段名占用11字节，加上type 1字节，length 4字节，使数据起始地址8字节对齐；

c) memory_map中每一条数据长度都是8字节倍数。

memory_map 内部结构是一个结构体数组，每一个结构体描述一段内存（下表中一个 region），数据结构见表 15。

表 15 Memory Map Region 数据结构

字段		偏移量（字节）	大小（字节）	说明
Region[0]	base	0	8	该区域的基地址，必须是系统物理地址，并且与 4K 对齐。
	Size	8	8	区域大小
	Type	16	1	请参阅下面的 Memory Type 定义。
	Reliability	17	1	请参阅下面的 Memory Reliability 定义。
	Hotplug	18	1	0：不支持热插拔 1：支持热插拔
	Reserved	19	5	保留字段
region[1]		24	24	区域 1
region[...]			24	区域 X
region[n]		24*n	24	区域 n

Memory 类型定义见表 16。

表 16 内存类型定义表

Memory Type	取值	说明
Free	0	未使用的内存类型
Data	1	它包含有效数据，读取数据后可用作空闲内存
Code	2	它包含 Receiver 的有效代码
Shared	3	用于在 Reporter 和 Receiver 之间共享信息，应按照惯例使用。
Reserved	4	不能使用，已被占用
Disabled	5	不能使用，可能有硬件故障
Device	6	来自设备的内存，不应添加到常规内存管理中，例如 MMIO、SRAM 等

注：Code：指的是属于接收者的代码，包括接收者自己的代码或者控制权属于接收者的代码，如 BIOS 加载 OS 相关代码到内存，并将控制权交接给 OS，则该地址区间便是 Code 类型，BIOS 自身驻留并且不直接供 OS 执行（即控制权不交接给 OS）的代码属于 Reserved 类型。

Memory 可靠性定义见表 17。

表 17 内存可靠性定义表

Memory Reliability	取值	说明
Normal	0	大多数 DDR 都是正常可靠性
High reliability	1	可靠性高于正常水平
Low reliability	-1	可靠性低于正常水平

注：

a) 可靠性是一个相对概念，因此在本文件中并不限制什么类型的内存定位什么可靠性级别。比如：

场景1：DDR大部分可以是normal类别，其中介质可靠性更好的可以是high类别，介质可靠性低的可以是low类别；

场景2：同样是DDR，实现mirror的可以是high，未实现的可以是normal。

b) 可靠性按照有符号整数解析，值越大可靠性越高，值越小可靠性越低。

11.3.4 Call ID Service 信息表

Call ID Service 信息表是用于 Call ID Service 提供者向调用者告知支持的 Call ID Service 范围及调用方式的信息表。

在同一个计算系统中，有多个组件可以提供 Call ID Service。Call ID Service 信息表可各不相同，由提供者自行维护，比如 BIOS 可以给 OS 和 BMC 提供 Call ID Service 信息表，外设可以给 BIOS 提供 Call ID Service 信息表，BMC 也可以给 BIOS 提供 Call ID Service 信息表，这些 Call ID Service 信息表内容可以各不相同，但信息表格式是一致的。另一方面，同一个组件面向不同组件提供的 Call ID Service 也可以不同，比如某个 BIOS 的接口可提供给 OS 调用，但不能提供给 BMC 调用，此时 BIOS 向 OS 传递的 Call ID Service 信息表可以包含这个接口，但提供给 BMC 的 Call ID Service 信息表不包含这个接口。在一个复杂的计算机系统内，可能存在复杂的接口调用关系，举例见图 20。

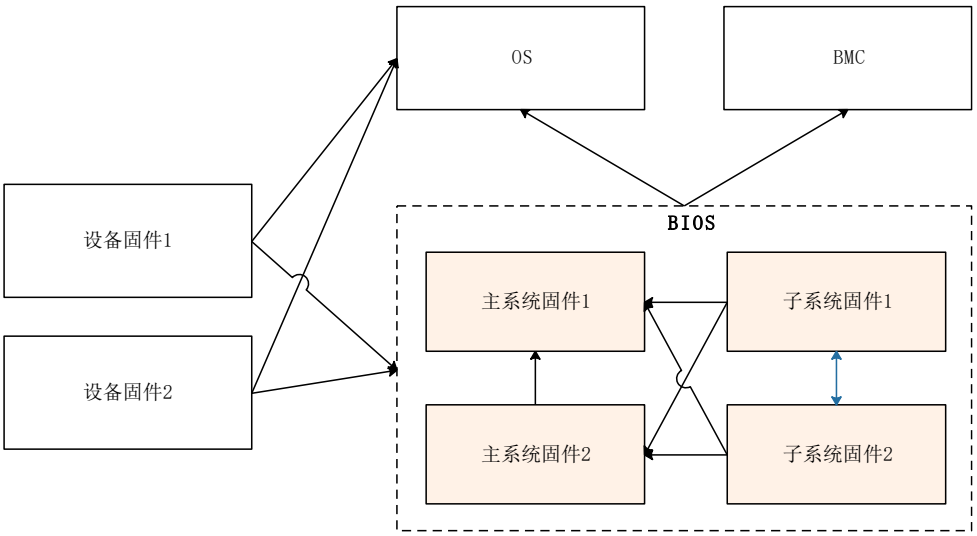

图 20　计算机系统内接口调用关系图

计算机系统内接口调用关系说明：

a) BIOS中多个子系统支持相同的Call ID：在BIOS中存在多个子系统，每个子系统都可以对外提供Call ID Service，并且可能存在2个子系统支持相同的Call ID的情况，且功能存在差异。

b) 相同型号的外设，支持相同的Call ID：上图2个device是型号、规格完全相同的，他们对外提供的完全相同的Call ID Service，区别只是设备的物理地址不同。

Call ID Service 信息表应能对上述各种结构进行描述。Call ID Service 信息表设计原则如下：

a) Call ID Service信息表内按照提供者User ID进行分组描述，描述本User ID对应组件所支持的所有Call ID Service；

b) Call ID所属的User ID分组按照最小组件的User ID分组；

注：对于每一个 Call ID Service，它所属 User ID 分组规则举例：

a) Call ID 1由子系统1响应，且在BIOS内唯一，则Call ID 1的User ID也应当为子系统1而不是范围更大的BIOS User ID；

b) Call ID 2可由子系统1和2响应，且功能不同，则Call ID 2应分别体现在系统1和2的User ID分组中；

c) Call ID 3可由子系统1和2响应，且功能相同，则Call ID 3可以分别体现在系统1和2的User ID分组中，也可以只体现在系统1或2的User ID分组中。

Call ID Service 信息表支持描述多个组件支持的 Call ID Service。每个组件支持的 ID 及其数量可以各不相同，每个组件设置一个或多个 group（通过 User ID 区分），每个 group 长度可以各不相同，数据结构之间无空隙，连续排布，见图 21。

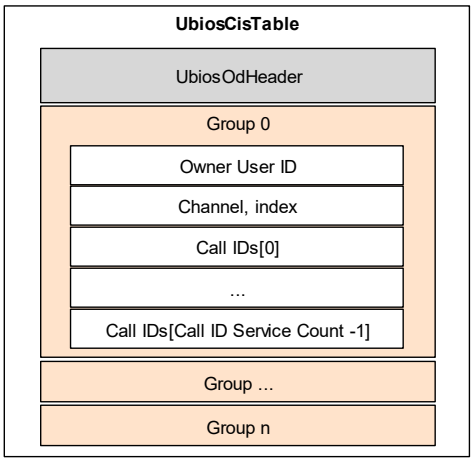

图 21 UBIOS CIS 信息表图示

注：

a) 需要保证每一组（group）内Call ID不存在重复；

b) UBIOS信息交互层发起Call ID Service调用时，从本信息表的起始开始遍历，找到第一个符合的Call ID，并将receiver User ID替换为此Call ID对应的owner User ID发起调用。即用户可以以更大范围的User ID作为receiver进行调用，此范围内可能存在多个支持此Call ID的子组件，那么以第一个（排在本信息表前面）支持此Call ID的组件作为目标receiver发起调用。

- Call ID Service对象描述：

```
ubios_ob_v1("call_id_service", total_size) {
    group@struct[group_number] = {
        owner@u32 = value
        usage@u8 = value
        index@u8 = value
        call_id@u32[call_id_service_count] = { value }
        forwarder@u32 = value
    }
    ub@struct = {
        usage@u8 = value
        index@u8 = value
        forwarder@u32 = value
    }
}
```

- group_number：表示本对象中 group 的数量，可省略。
- call_id_service_count：表示 Call ID Service 数量，可省略。

注：value 仅表示此处填值，没有特指，只是示意表达。

CIS 对象描述头结构体描述见表 18。

表 18 Call ID Service 对象描述头(Object Description Header)结构体

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“call_id_service”
Total Size	16	4	整个表的总大小（包括此表头）

字段	偏移量（字节）	大小（字节）	说明
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 注： a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

CIS 对象描述成员列表见表 19。

表 19 Call ID Service 对象描述成员列表

名称	类型	是否必选	说明
group	struct list	是	描述由具有独立 User ID 的一个组件支持的每个 Call ID Service，一个组件一组，组的定义见下文
ub	Struct	否	描述用于通过 UB 调用其他 UBPU 的通道信息

group 是一个结构体列表，列表的每个成员是一个结构体，结构体中成员见表 20。

表 20 group 结构体成员说明表

名称	类型	是否必选	说明
owner	u32	是	该组所属组件的 User ID
usage	u8	是	0：不支持，尚未使用，将来可用于动态禁用 Call ID 服务 1：使用 ISA 调用 2：使用 UVB 3：保留 其他值：未定义，通常由设备固件使用
index	u8	否	当相同 usage 的项数大于 1 时，需要指定要使用的具体 usage。例如： 如果使用的是 UVB，则表示“12.2.4 UVB 信息表”中 UVB 的索引
call_id	u32 list	是	与 Call ID 定义相同，指示该组件支持哪个 Call ID
forwarder	u32	否	组件的 User ID，指示特定组件转发消息

注：Call ID Service 信息表中只需要描述当前组件支持的 Call ID，不支持的可以不描述。

ub 是一个结构体，结构体成员见表 21。

表 21 ub 结构体成员说明表

名称	类型	是否必选	说明
usage	u8	是	见表 20 group 结构体成员说明表
index	u8	否	见表 20 group 结构体成员说明表
forwarder	u32	否	见表 20 group 结构体成员说明表

11.3.5 UVB 信息表

UVB 信息表用于 BIOS 启动时向 OS 上报。若无说明，本节中虚拟总线或 UVB 的描述即表示内存虚拟总线。

UVB 信息表是 BIOS 向 OS 传递内存虚拟总线规模及位置的核心机制。BIOS 完成内存虚拟总线初始化后生成该表，明确划分虚拟总线窗口空间，OS 基于表中信息进行使用资源调度和使用。

UVB 信息表仅支持共享内存方式传递。使用临时性共享内存空间进行主动上报为必备，使用 Call ID Service 获取方式为可选。

UVB 信息表支持描述多条不同的 UVB，每条 UVB 的长度可以不同，UVB 数据结构之间无空隙，连续排布，见图 22。

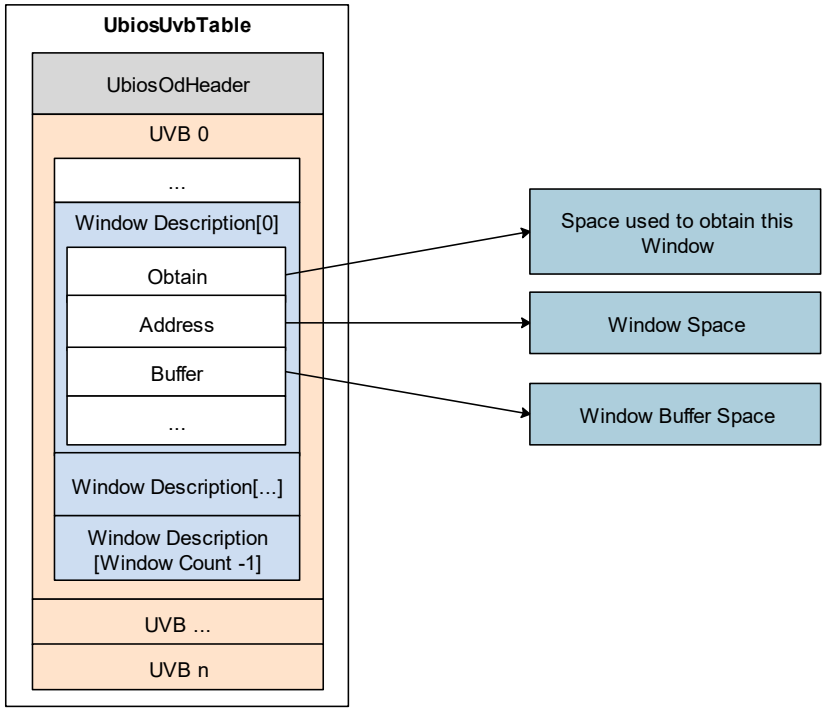

图 22 UVB 信息表窗口图

UVB 信息表可以包含多条虚拟总线信息（UVB），每条虚拟总线（UVB）可以包含多个窗口，每个窗口由一个窗口描述信息（Window Description）描述，窗口描述信息中包含窗口的占用地址、窗口的地址、窗口的专用缓存及缓存长度信息。

窗口的专用缓存是可选的（如果存在则必须使用），用于解决通信彼此不能全局访问彼此的空间的情况，有利于提升安全性。如果指定了缓存空间，双方交互的所有数据（如 Input Data Address 和 Output Data Address）必须在该缓存空间内，不能越界。

注：

a) 左侧UVB信息表内容为静态信息；

b) 右侧为虚拟总线操作的地址空间，使用时会对该空间进行读写操作，不直接包含在表中。

UVB对象描述：

```
ubios_ob_v1("virtual_bus", total_size) {
    uvb@struct[uvb_number] = {
        delay@u32 = value
        wd@table[window_count]:
```

```
            obtain@u64, address@u64, buffer@u64, size@u32 = {
            value
        }
    }
}
```

注：value 仅表示此处填值，没有特指，只是示意表达。

UVB 对象描述头说明见表 22。

表 22 UVB 对象描述头说明

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“virtual_bus”
Total Size	16	4	整个表的总大小（包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 **注：** a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

UVB 对象描述成员见表 23。

表 23 UVB 对象描述成员列表

名称	类型	是否必选	说明
uvb	struct list	是	每个 UVB 的信息，见下文

uvb 是一个结构体列表，列表的每个成员是一个结构体，结构体中成员见表 24。

表 24 uvb 结构体成员表

名称	类型	是否必选	说明
delay	u32	否	在获取窗口时读回之前写入 User ID 后延迟 ms
Wd	table	是	Windows 描述，一个窗口有一个 Windows 描述，定义见下文

注：当 delay 不为 0 时，同时需要在抢占窗口步骤 1~3 时关中断，以防被中断打断导致 delay 延时不足。

wd 是一个表格，表格的成员见表 25。

表 25 wd 成员说明

名称	类型	是否必选	说明
obtain	u64	是	用于获取该窗口的系统物理地址（往该地址写占用者 User ID）
address	u64	是	UVB 窗口的系统物理地址。窗口区域必须是内存映射中的 shared 类型
buffer	u64	否	该窗口的特定空间的系统物理地址（如果存在）。该空间必须是内存映射中的 shared 类型
size	u32	否	缓冲区的大小。 如果存在缓冲区，则这是必需的。 如果缓冲区不存在或缓冲区存在且为 0，则忽略。

注：

a) 本文件中约定 table 类型的成员顺序是确定的，不可随意变化（下同）。

b) 当 buffer 存在且不为 0 时，UVB 应通过该 buffer 来传递 input 和 output 信息。input 和 output 共用同一 buffer， input 数据和 output 数据应依次放置，起始地址应按 8 字节对齐。。

c) UVB窗口的设置需保证所有此UVB的使用方具有cache一致性（不开cache也是一种特殊的一致性），范围包括UVB窗口及其对应的buffer。

11.3.6 芯片 ISA 交互信息表

在当前文件版本中，芯片 ISA 交互指通过芯片专用指令实现 ISA Call（即 Call ID Service 信息表中 usage 为 1 的方式）。

芯片 ISA 交互会涉及到芯片权限级别的切换，存在一定的安全风险，尤其是通过共享内存进行参数数据传递时，由于没有限制范围，可能存在提权读写的情况，为了消减这一风险，可以通过本信息表约定通过芯片 ISA 交互时的共享内存范围，由信息交互层负责适配和检查参数数据传递的安全性。

ISA Call 对象描述：

```
ubios_ob_v1("isa_call", total_size) {
    buffer_type@8 = value
    buffer_index@8 = value
}
```

ISA call 对象描述头结构见表 26。

表 26 ISA call 对象描述头结构表

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“isa_call”
Total Size	16	4	整个表的总大小（包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 注： a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

ISA Call 对象描述成员列表见表 27。

表 27 ISA Call 对象描述成员列表

名称	类型	是否必选	说明
buffer_type	u8	否	指示在 ISA 调用中使用缓冲区传输数据时的缓冲区类型。 2：UVB 其他值：保留
buffer_index	u8	否	缓冲区的索引，与 call_id_service 的 group@struct 中的索引含义相同。

注：

a) 使用buffer传递数据只是风险消减措施之一，并不能完全消除风险，由于信息交互层不会解析参数内容，只能保证参数传递的安全性，但不能保证参数中可能存在的非法地址访问，例如参数中含有一个指向非法地址的指针，ISA Call响应程序使用该指针进行数据读写，这一风险需要由各响应程序自行消减。

b) 如果不存在此信息表，则信息交互层不会进行内存范围适配和检查，需由使用者自行保证。

11.3.7 Notify 信息表

通过 Notify 信息表来向接收者描述基于共享内存传递 Notify ID Information 的方式，非共享内存传递无需额外描述。

当前可用于传递 Notify ID Information 的共享内存方式有 2 种：

a) UVB 方式：是阻塞式、带反馈的，可用于传递重要、必须等到接收者应答的消息。可以复用 UVB 的功能，详见 UVB 章节。

b) ring buffer 方式：是异步、无反馈的，发送者只写，接收者只读，buffer 中的消息先进先出，可用于传递无需接收者应答的消息，尤其较频繁的消息。

同一时刻 ring buffer 中可能会存在多个 Notify ID Information，每个 Notify ID Information 都有自己的数据结构定义，可能长度各不相同，本文件约定 ring buffer 中各个 Notify ID Information 连续排布，中间不留空隙，即一个 Notify ID Information 的结尾是下一个 Notify ID Information 的开始。

单个 ring buffer 的 Notify 信息表见图 23。

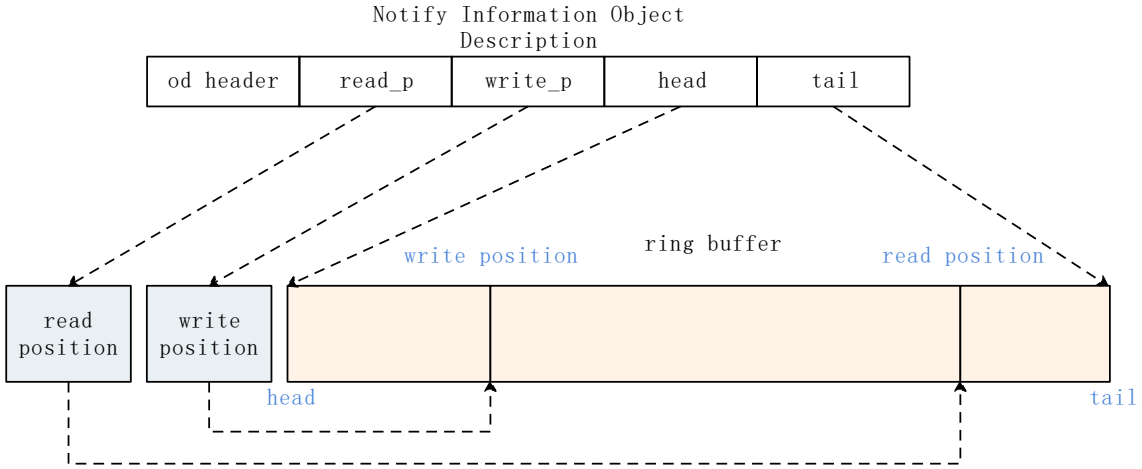

图 23 Ring Buffer 的 Notify 消息表图

注：

a) Read position、write position、ring buffer这3者所占的空间可以不连续，每一个的地址都需要满足对齐访问要求，即8字节对齐。

b) Notify信息表的内容是不随Notify ID Information的读写而改变的。

Notify信息表对象描述：

```
ubios_ob_v1("notify_info", total_size) {
    ring_buffer@struct = {
          head@u64 = value
          tail@u64 = value
           read_p@u64 = value
```

```
            write_p@u64 = value
        }
        uvb_index@u8 = value
        irq@struct = {
            number@u32 = value
            clear@raw = {
                @u64 = interrupt register address
                @u64 = interrupt clear mask
                @u64 = interrupt clear write value
            }
        }
    }
```

注：value 仅表示此处填值，没有特指，只是示意表达。

Notify 信息表对象描述头结构见表 28。

表 28 Notify 信息表对象描述头结构

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“notify_info”
Total Size	16	4	整个表的总大小(包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 **注：** a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

Notify 信息表对象描述成员列表见表 29。

表 29 Notify 信息表对象描述成员列表

名称	类型	是否必选	说明
ring_buffer	struct	否	环形缓冲区的信息，见下文。
uvb_index	u8	否	UVB 在 virtual_bus 中的索引，用于传输 Notify ID 信息。
irq	struct	否	中断请求信息，见下文。

注：建议用于传递 Notify ID Information 的 UVB 与用于执行 Call ID Service 的 UVB 分开。

ring_buffer 是一个结构体，结构体中成员见表 30。

表 30 ring_buffer 结构体成员说明表

名称	类型	是否必选	说明
head	u64	是	环形缓冲区的起始位置，用于传输 Notify ID 信息，是系统物理地址。
tail	u64	是	用于传输 Notify ID 信息的环形缓冲区的末尾，是系统物理地址。

名称	类型	是否必选	说明
read_p	u64	是	存储读取位置的地址，表示有效 Notify ID Information 的开始。
write_p	u64	是	存放写入位置的地址，表示有效 Notify ID Information 的结束，也是空区域的开始。

注：上述地址皆按 64 位表达，对于 32 位系统可忽略高 32 位。

发送者和接收者可以同时操作同一个 ring buffer，无需进行独占操作，由接收者负责修改 read position，由发送者负责修改 write position，接收者需要读取到完整的 Notify ID Information 才能修改 read position 的值，发送者需要写入完整的 Notify ID Information 才能修改 write position，如果写入时发现空间不足可以放弃写入或延迟写入，具体由实现决定。

ring_buffer 中的数据格式见表 31。

表 31 ring_buffer 数据格式表

字段	偏移量（字节）	大小（字节）	说明
Notify ID	0	4	Notify ID
Data Length	4	4	不包括 Notify ID 和数据长度字段。 0 表示没有数据。
Data	8	Data Length	数据，由 Information ID 决定.

注：

a) Data中真实的Information数据长度可能不是8字节的倍数，不保证各个信息按8字节对齐存放，读写操作时需注意对齐访问问题。

b) Notify信息表是单向的，即只能由一方向另一方报告，如果存在双向的需求，应当设置2个独立的Notify信息表分别由发送者提供给接收者。

c) Notify信息表及read position、write position、ring buffer所在内存应当在Memory Map中体现，且Memory Type为Shared。

irq 是一个数据结构体，结构体成员描述见表 32。

表 32 irq 数据格式表

名称	类型	是否必选	说明
number	u32	是	用于向 OS 发送信号的中断号
clear	raw	是	用于清除中断状态的信息。

Clear 结构定义如下：

```
struct IrqClear {
   u64 addr;
   u64 mask;
   u64 value;
};
```

clear 用法示例：

```
{
   data = read(addr);
   data = (data & mask) | value;
   write(data);
}
```

注：

a) 按实际位宽操作，如果 32 位系统，忽略高位。
b) Notify 信息表是单向的，即只能由一方向另一方报告，如果存在双向的需求，应当设置 2 个独立的 Notify 信息表分别由发送者提供给接收者。
c) Notify 信息表及 read position、write position、ring buffer 所在内存应当在 Memory Map 中体现，且 Memory Type 为 Shared。

11.4 其他信息表引用

其他信息表特指由其他规范定义的信息表，例如 device tree，此处描述其他信息表如何引用到 UBIOS 信息表中。

其他信息表的引用也采用 UBIOS 对象描述格式，统一纳入名为 other_tables 的对象描述中。引用通常有 2 种方式：

a) 地址引用：指在other_tables中只包含一个对应信息表的地址，对应信息表内容的变化不影响 other_tables；
b) 文件引用：指将对应信息表内容完全添加到other_tables中，对应信息表的任意修改都会影响 other_tables。

注：优选地址引用，其他信息表通常有它自己的格式检查，不需要依赖 UBIOS 对象描述的格式检查。

Other Tables对象描述：

```
ubios_od_v1("other_tables", total_size) {
    dtb_address@u64 = value
}
```

注：value 仅表示此处填值，没有特指，只是示意表达。

其他信息表对象描述头结构说明见表 33。

表 33 其他对象描述头结构说明

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名，“other_tables”
Total Size	16	4	整个表的总大小（包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 注： a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

Other Tables 对象描述成员列表见表 34。

表 34 其他信息表对象描述成员

名称	类型	是否必选	说明
dtb_address	u64	否	设备树 blob 的系统物理地址

附录 A

（规范性附录）
UBIOS 对象描述

A.1 目的

为了匹配 UBIOS 规范应用场景和发展演进需求，本文设计定义了一套简单易用、资源需求小、可扩展性高、可靠性高的数据描述格式，称为 UBIOS 对象描述规范（下简称本文件）。

本文件不仅可用于组件间信息传递，还可用于数据代码分离设计所需的静态配置。

本文件只定义对象描述的规则和语法，不定义具体对象的内容。

A.2 概述

本文件定义了一种 name/value 结构的信息描述性语言及其对应的二进制数据格式，用来描述 UBIOS 对象信息。

A.2.1 总体数据结构

UBIOS 对象描述分为 2 层：对象描述根（od root）和对象描述文件（od file）。其结构示意如下：

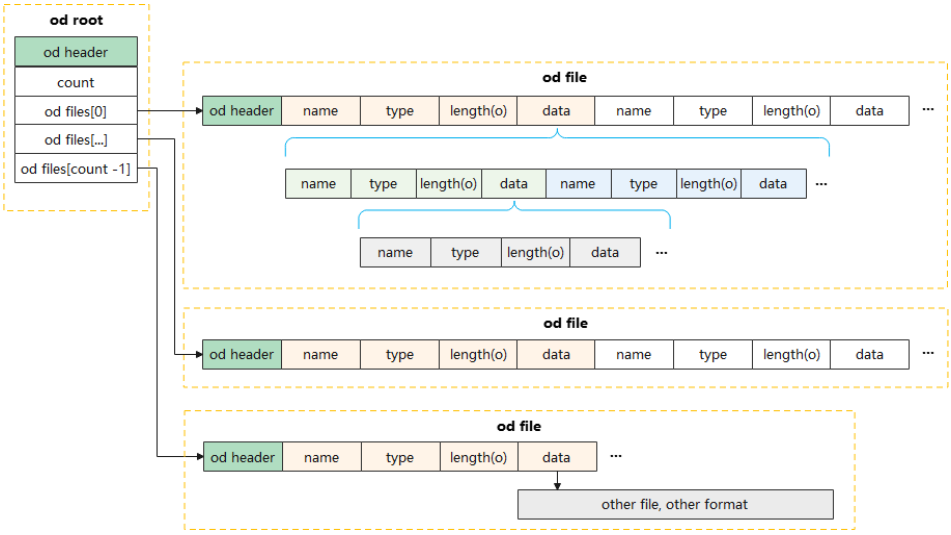

图 A.1 UBIOS 对象描述层级图

od root：负责将所有对象描述文件组合在一起，od root作为UBIOS对象描述的信息入口，通过od root可以搜索到所有对象描述文件，od root起始地址8字节对齐。

od file：以对象为单位，一个对象对应一个对象描述文件，每个文件相互独立，每个文件内容由1个od header与不定量的name/value信息组成，od file起始地址8字节对齐，内容按1字节对齐连续存放。

注：

a) 只允许od root中嵌入对象描述文件，对象描述文件之间不能嵌套。

b) od file可以嵌入其他文件，详见值描述中文件章节。

c) od tools打包od file时，如果前一个od file结束地址非8字节对齐，则需跳到下一个8字节对齐地址开始存放当前的od file，同时不改变前一个od file的长度。

name/value数据格式：采用TT[L]V（tag、type、length、value）的编码格式，每个name/value的存储数据格式分为name、type、length、data 4部分，data中可嵌套其他name/value数据，无嵌套层数限制，各部分含义如下：

表 A.1 name/value 字段说明

字段	大小（字节）	说明
name	不固定	见 name 定义
type	1	见 type 定义
length	4	data 大小，不含 name、type、length，当 data 长度固定时省略
data	不固定	见各种值描述定义

注：本文件所描述的 UBIOS 对象描述，无特别说明的情况下均采用小端格式，数据大小均以字节为单位。

A.2.2 文件定义及其关系

UBIOS 对象描述采用文本化编写，通过工具编译成二进制文件供程序使用，具体实现可参考以下方式，本文件不强制要求。

文件分为 3 种类型，分别是：

a) **.ods:** object description source file

人工编写的源码文件

b) **.odsi:** object description source intermediate file

基于.ods 预处理后生成，包括引用文件合并、宏展开、表达式计算、错误检查等操作之后生成的中间文件

c) **.od:** object description file

基于.odsi 转换而成的最终的二进制对象描述文件

ods 文件通过 od tools 转换成 odsi 文件，再通过 od tools 转换成 od 文件，3 种文件及 od tools 的关系如下：

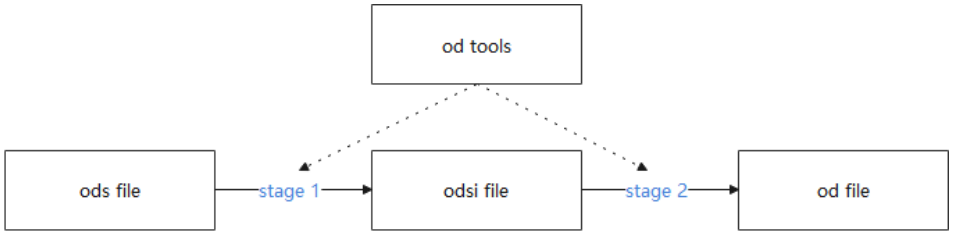

图 A.2 UBIOS 对象描述文件关系图

A.3 语法定义

A.3.1 基础语法格式

```
ubios_od_root_version(root_name, max_od_number) {
     ubios_od_version(object_name, total_size) {
            name@type = value
       }
}
```

大括号{}：表示层级关系和范围，{}括起来的内容在上一层被看成一个value

换行：表示一个value的结束，一行最多只能写一条name/value信息

空格：除特别说明外无作用

逗号,：参数或value中多个成员的分割符，如果一行的最后是逗号，则表示换行且未结束，下一行仍然是参数或value中的成员

@：命令符，name后的@表示指定name对应value的数据类型

等号=：表示赋值

注：

a) 标点符号需是半角。

b) 如果一个value内部又由多个子value组成，并且想要分成多行书写，则需要在外层加上大括号{}。

值名称 (Name)

name：不需要""，不区分大小写，最终全小写保存，支持的字符如下表：

表 A.2 name 支持字符

类型	字符
字母	a ~ z 和 A ~ Z
数字	0 ~ 9
符号	下划线_

注：

a) name最长16字节，含结束符。

b) 同层name不允许重名，不同层无约束。

值类型 (Type)

type：准确描述name对应的值类型，@type 可以省略，省略则根据value确定默认类型，type在转换成二进制时会被记录，以便在取值时判断类型是否匹配，提升错误可定位行，类型定义如下表：

表 A.3 type 类型

type	含义	对应的类型值	数据长度（字节）	使用举例
u8	无符号 8 位整数	0x1	1	name@u8
u16	无符号 16 位整数	0x2	2	name@u16
u32	无符号 32 位整数	0x3	4	name@u32
u64	无符号 64 位整数	0x4	8	name@u64
i8	有符号 8 位整数	0x5	1	name@i8
i16	有符号 16 位整数	0x6	2	name@i16

type	含义	对应的类型值	数据长度（字节）	使用举例
i32	有符号 32 位整数	0x7	4	name@i32
i64	有符号 64 位整数	0x8	8	name@i64
bool	布尔类型	0x10	1	name@bool
char	单个字符	0x20	1	name@char
string	字符串	0x21	字符串长度，含结束符	name@string
struct	结构体	0x30	不固定	name@struct
table	表	0x40	不固定	name@table
raw	原始数据	0x50	不固定	name@raw
无	列表	0x80 + 类型值	不固定	无

注：列表是在原有数据类型基础上增加的信息，以原类型值加 0x80 作为列表对应的类型值。

A.3.2 声明

声明对象描述根 (Object description root)

1. **语法：**ubios_od_root_version(root_name, max_od_number) { }
2. **解释：**声明（或创建）UBIOS对象描述的od root，作为最终输出的UBIOS对象描述信息的起始，整个UBIOS对象描述信息中必须有且只能有一个。
   - ubios_od_root ：固定写法
   - _version ：表示od root的版本，v+数字表示，例如ubios_od_root_v1表示版本为1的od root，可省略，省略等同于v1
   - ( )：固定写法
   - root_name ：字符串，对象根名称，规则与name相同
   - max_od_number ：无符号整数，可省略，表示当前od root中最多可包含的对象数量，实际数量根据实际内容计算得到，如果省略则值为实际数量，如果未省略且值小于实际数量，则od tools需要报错
   - { }：固定写法

   只能有一个 ods 文件包含此内容，其他 ods 文件只需要声明 ubios_od 即可。

   对应数据结构：

   ubios_od_root_v1 数据结构下表：

表 A.4 ubios_od_root_v1 数据结构

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	表名
Total Size	16	4	整个表的总大小包（括此标题）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小

字段	偏移量（字节）	大小（字节）	说明
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 **注：** a) 有效数据指从本表 header 开始，大小为 Total Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。
Count	32	2	max_od_number 或实际 od file 数量
Reserved	34	6	Reserved
OD Files[Count]	40	8*Count	od file 的系统物理地址，每个 od file 占 1 个位置，为 0 表示此处为空，未指向任何一个 od file

注：

a) od tools生成od file时，od file的地址为基于od root起始地址的偏移，程序加载od root和od file时需更新此od file地址的值。

b) OD Files[Count]中有效值可不连续存放，即可存在空隙，值为0表示空隙，查询时应当遍历。

声明对象 (Object description)

语法：ubios_od_version(object_name, total_size) { }

解释：声明（或创建）一个UBIOS对象描述。

- ubios_od ：固定写法
- _version ： 与od root定义相同，表示od file的版本
- ( )：固定写法
- object_name ： 字符串，对象名称，规则与name相同
- total_size ： 无符号整数，可省略，表示当前对象文件占用的总大小（含od header），实际大小根据实际内容计算获得，如果省略则值为实际大小，如果未省略且值小于实际大小，则od tools需要报错
- { }：固定写法

对应数据结构：

图 A.3 对象数据结构

od header：ubios_od_v1 数据结构如下

表 A.5 ubios_od_v1 数据结构

字段	偏移量（字节）	大小（字节）	说明
Name	0	16	object_name ，固定 16 字节长度，含结束符
Total Size	16	4	整个表的总大小（包括此表头）
Version	20	1	版本号
Reserved	21	3	保留字段
Remaining Size	24	4	本表剩余空间大小
Checksum	28	4	校验和填充，填充后使有效数据以 4 字节不进位累加，结果为 0 表示数据正确，否则表示数据存在错误 **注：** a) 有效数据指从本表 header 开始，大小为 Total

字段	偏移量（字节）	大小（字节）	说明
			Size - Remaining Size 的数据区域； b) 若结尾不足 4 字节，则高位补 0 后再进行累加计算。

data：大括号内内容，大小为 od header 中的 Total Size 字段。

注：

a) 对象的name固定16字节，不足部分补0，与值的name长度有区别。

b) 整个od file大小可能不是4字节倍数，计算Checksum时不足4字节时补0。

声明文件引用 (File import)

语法：@import path/file.ods

解释：声明一个文件引用，在当前位置引进一个ods file。不同的对象描述可以写在同一个文件中，也可以各个对象分开书写，分开写时，可以通过此命令将各描述文件添加到od root范围内，最终形成完整的对象描述文件。

- @import：文件引用关键字
- 空格：空格将字段分隔开
- path/file.ods：对象文件路径和名称，基于当前位置的相对路径

注：当前只支持引用.ods 文件，只支持在 root 层插入对象描述，对象描述中不支持插入其他对象描述。此处空格是必备的，不能忽略。

声明继承首个值（Inherit first value）

语法：

```
@inherit  name@struct[number] =
       {name1 = value1, name2 = value2},
         {name2 = value3},
         {name3 = value4}
```

解释：申明在书写一个值时，自动继承它的首个值，并且可以在继承的基础上按需修改和扩展，减少书写冗余。

- @inherit：声明继承的关键字
- 空格：空格将字段分隔开
- {name2 = value3}：
  - 继承首个值，并修改其中名为name2的值，等价于第二个值为{name1 = value1, name2 = value3}
  - 支持同时修改多个成员的值，通过,或换行隔开，与正常的多值书写格式一致
  - 支持修改指定成员的下级成员的值，多个name用.连接指定，如name_a.name_b = value
  - 如果只有{}，表示复制第一个值不做修改
- {name3 = value4}：因首值中没有name3，所以在继承首个值的基础上增加name3 = value4，属于扩展

注：当前@inherit 只支持修饰 struct list 类型的值，且不支持删除首值中的成员。

声明宏定义 (Define Macro)

语法：@define a b

解释：声明一个宏，解析时执行文本替代，将a替换为b，只作用于本文件中的value内。

- @define：宏定义关键字
- a：字母、下划线、数字组成的字符串，长度无限制，不支持参数

- b：内容参照value
- 空格：空格将字段分隔开

注：

a) 宏的替代只作用于value内，应避免a与任何非value字符重叠，建议a中字母以大写编写，而非value中字母均以小写编写。

b) 宏只作用于宏定义之后的内容，不可以跨文件生效，即当被引用文件中包含宏定义时，该宏定义只影响该被引用文件，不影响引用它的文件，但是引用文件处的宏定义会影响被引用文件，因此，如果希望宏全局生效，应当声明在od root文件中或通过在编译工具od tools增加-D MACRO的方式定义。

c) 当多个文件存在相同的宏时，如果值相同则没有问题，如果值不同则会报错。

d) 此处空格是必备的，不能忽略。

声明引用宏定义 (Include Macro)

语法：@include path/file.h { MACRO }

解释：声明引用一个第三方文件中的宏定义。

- @include：引用宏定义的关键字
- 空格：空格将字段分隔开
- path/file.h：第三方文件路径和名称，基于当前位置的相对路径

注：当前只支持引用C语言的头文件.h文件。

- {}：用于指定需要引用的宏名称
- MACRO：需要从文件中获取的宏定义名称，支持同时写多个，通过逗号或换行分隔

注：

a) 本文件有自己的宏定义使用规则，基本上可认为它是C语言宏语法的子集，不是全集，所以通过@include引用时，可能存在不符合本文件语法的宏，如果该宏被使用，则会在od tool执行中报错，如果该宏未被使用，则不会有影响。

b) 不支持嵌套，即od tool只解析path/file.h内的宏，忽略它所包含的其他文件。

A.3.3 值描述

整数 (Integer)

语法：name@type = 123或name@type = 0x1234 5678

解释：value支持十进制和十六进制表达方式，十六进制采用0x开头，支持表达式，由od tools负责计算并保存最终结果，支持的表达式算法详见表达式章节。

注：如果未指定 type，则默认类型为无符号 64 位整数（即 u64）；如果指定了 type，则 od tools 需要检查 value 与 type 是否匹配，不匹配需报错。

对应数据结构：

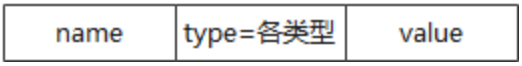

图 A.4 整数数据结构

布尔数 (Boolean number)

语法1：name@bool = 0或1

解释：布尔类型必须指定type为bool，value支持表达式，支持的表达式算法详见表达式章节。

0 表示 false

1 表示 true

注：如果未指定类型，值 0 或 1 默认类型是无符号 64 位整数。

语法2： name = false 或 true

解释： false 或 true作为特定布尔值，不带双引号，od file中保存的仍然是0或1。

对应数据结构：

name	type=bool	0或1

图 A.5 布尔数数据结构

字符 (Character)

语法： name@type ='c'

解释： value为通过单引号"括起来的单个字符，此时type必须为char或省略。

对应数据结构：

name	type=char	'c'

图 A.6 字符数据结构

字符串 (String)

语法： name@type ="string"

解释： value为通过双引号""括起来的1个以上字符，转换成二进制后，值包含结束符，此时type必须为string或省略。

对应数据结构：

name	type=string	"string"

图 A.7 字符串据结构

列表 (List)

语法： name@type[number] = value1, value2

解释： 一组相同类型的值集合。

- @type：指定value的类型，可以省略，所有value类型必须一致，但长度不一定相同
  注：当前只支持整数、布尔数、字符、字符串、结构体，不支持其他类型。
- [ ]：表示列表，不能省略
- number：表示列表成员个数，可以省略，如果省略则是value的实际数量，如果number未省略且值小于实际value数量，则od tools需要报错，如果值大于实际value数量，则区分value类型处理，见下面说明

注：

a) 如果value类型为整数、布尔数、字符，则列表的每个成员长度一致，当number大于实际value数量时，仍然会占用number数量的空间，无value的位置补0；

b) 如果value类型为字符串、结构体，则列表的每个成员长度可能不一致，最终number值为实际字符串或结构体的数量。

对应数据结构：

name	type=list	length	number	value1	value2

图 A.8 列表数据结构

number： 表示列表成员个数，长度 4 字节。

注：如果 value 是结构体，那么它的数据结构是结构体 (Structure)数据结构从 length 开始的部分，包含 length。

结构体 (Structure)

语法：

```
name1@struct = {
    name2@type = value2
    name3@type[number] = value31, value32
    name4@struct = {
        name5@type = value5
    }
}
```

解释：结构体就是描述信息的嵌套组合，结构体可以嵌套结构体。

- @struct：通常会被省略，但也可以不省略以供od tools检查，避免书写错误
- { }：表示结构体的范围，不可省略

对应数据结构：

name1	type=struct	length			
name2	type	length(o)	value2		
name3	type=list	length	number	value31	value32
name4	type=struct	length			
name5	type	length(o)	value5		

图 A.9 结构体数据结构

注：忽略换行，上述数据连续无空隙排布，本文其他数据也类似。

表 (Table)

语法：

```
name1@table [max_row_number] : name2@type, name3@type = {
    value11, value12
    value21, value22
    value31, value32
}
```

解释：表是一种特殊的结构，在存在大量重复描述的时候，可以简化书写方式以及减少在二进制文件中的空间占用。

- @table：可以省略，但也可以不省略以供od tools检查，避免书写错误
- [max_row_number]：表示value的行数，主要用于检查，整体可以省略
- 如果max_row_number省略则保存value的实际行数
- 如果max_row_number未省略且值小于实际value行数，则od tools需要报错
- 如果max_row_number未省略且值大于实际value行数，则等价于省略
- :：表示后面是一串name@type，大括号{ }中每一行的value的顺序与此name@type顺序一一对应，不能省略
- name@type：当前版本type只支持整数、布尔数、字符类型，name不能省略，@type同基础值描述，可以省略

- { }：表示表的范围，不可省略

对应数据结构：

name1	type=table	length	row number		
column number	name2	type	name3	type	
value11	value12				
value21	value22				
value31	value32				

图 A.10 表数据结构

row number：数据行数，以一组 sub name 作为一行，长度 2 字节。

column number：即上述:后 name 的个数，表示 table 中子一层 name 的数量，长度 1 字节。

原始数据 (Raw)

语法1：

```
name @raw = {
    @type = value
}
```

解释：将不同类型的数据按1字节对齐方式排列作为name的值。

- @type = value：在其他值描述的基础上省略了name，其他与其他值描述相同，u64类型可省略@type和= ，多个值之间可以用,或换行分隔
- { }：表示范围，不可省略，大括号内部可换行

注：raw 语法 1 中不支持变长的数据类型，例如 string、struct。

语法2：name @raw = path/data_file

解释：将data_file原封不动打包进最终生成的.od文件。

- @raw：不可省略
- path/data_file：对应的原始数据路径和名称，不加""，路径是基于当前位置的相对路径

对应数据结构：

name	type=raw	length	data

图 A.11 原始数据数据结构

A.3.4 注释

语法：//注释内容

解释：每一行//之后的内容为注释信息，可以在行首，也可以在有效信息之后

A.3.5 表达式

表达式说明如下表：

表 A.6 表达式说明表

运算符	说明

运算类型	具体支持的算法符号
算数运算	加+ 减- 乘* 除/ 取余%
移位运算	左移<< 右移>>
位运算	与& 或| 异或^ 取反~
逻辑运算	与&& 或|| 非!
优先级	小括号()

A.3.6 宏编译

语法：

```
@ifdef  MACRO
    内容a
@else
    内容b
@endif
```

解释：根据条件MACRO是否为宏定义选择编译进od file的内容。

- @ifdef：判断MACRO是否是现存的宏，是则编译内容a，否则编译内容b，不可省略
- MACRO：用于判断是否为宏的符号，不可省略
- @else：表示前面的条件分支不成立时选择@else后面的分支，此分支不含条件，可省略

@endif：一个编译宏的结束，不可省略

附录 B

（参考性附录）
接口应用逻辑示例

B.1 概述

UBIOS接口在典型场景下的应用逻辑，分别以BIOS与OS、BIOS与BMC、BIOS与外设场景描述UBIOS接口是如何工作的，此处重点描述功能逻辑，以便读者理解UBIOS接口的工作原理，忽略各组件内部细节，包括信息交互层的处理、交互通道处理等。

B.2 BIOS 与 OS 间接口应用

B.2.1 概述

运行在同一个系统中的BIOS和OS宜通过共享空间进行数据传递。共享空间可以是内存（DDR、HBM、SRAM等）、寄存器（通用寄存器、芯片寄存器等）等。

B.2.2 Call ID Service 应用场景

BIOS 与 OS 间 Call ID Service 接口是单向的，只能 OS 调用 BIOS，可用 UVB 和芯片 ISA 专用指令实现调用。

a) UVB方式调用

OS 通过 UVB 调用 BIOS 的 Call ID Service 典型过程见图 B. 1。

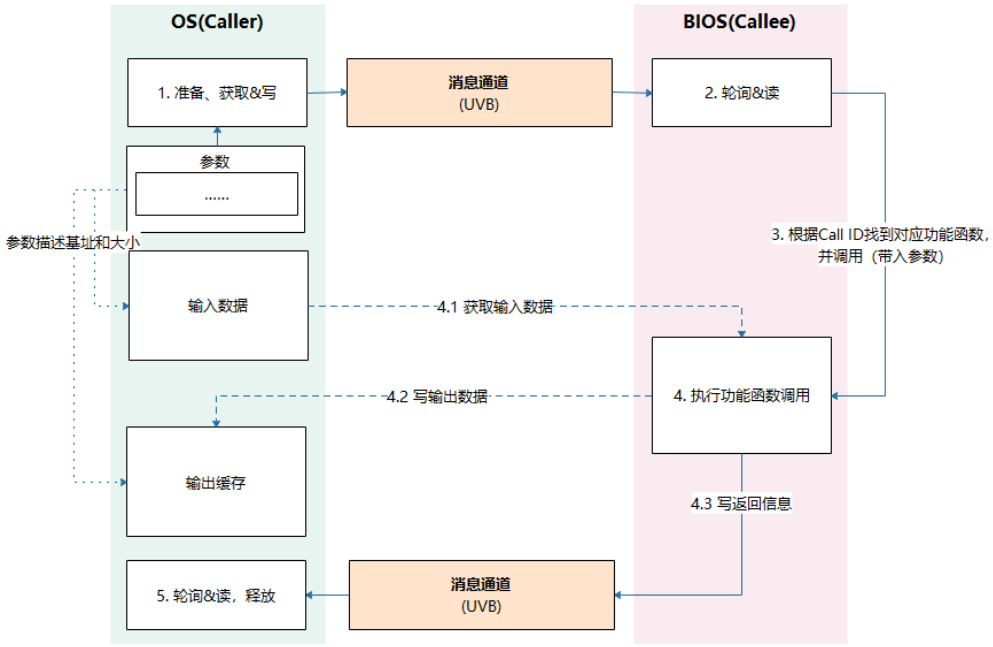

图 B. 1 OS 通过 UVB 调用 BIOS 的 Call ID Service 典型过程

采用 UVB 方式调用步骤描述如下：

步骤 1：Caller（OS）准备好入参，并占用一个 UVB 窗口，将目标功能的 Call ID 和参数信息按 UVB 窗口格式要求写入窗口；

步骤 2：Callee（BIOS）轮询 UVB 窗口，获得 Call ID 和参数信息；

步骤 3：Callee（BIOS）根据 Call ID 定位到具体功能，并将参数信息传入，执行该函数；

步骤 4：具体功能通过参数信息定位到 Caller 存放入参的区域，并读取入参，执行功能，将结果写入 Caller 指定的接收地址，并通过同一个 UVB 窗口返回执行结果信息；

步骤 5：Caller（OS）轮询本次用于发起调用的 UVB 窗口，获取执行结果，并释放 UVB 窗口。

注：上图步骤 4.1、4.2 虚线表示可能经过信息交互层转存，详见信息交互层描述。

b) 芯片ISA专用指令方式调用

参考各芯片ISA手册。

B.2.3 Notify ID Information 应用场景

BIOS 与 OS 间的 Notify ID Information 是单向接口，只允许 BIOS 通过 UVB 或通过中断方式向 OS 上报。两者主要差异在于 UVB 需要指定 Notify ID，而中断可以不指定。

a) UVB方式上报

采用 UVB 上报的典型过程见图 B.2。

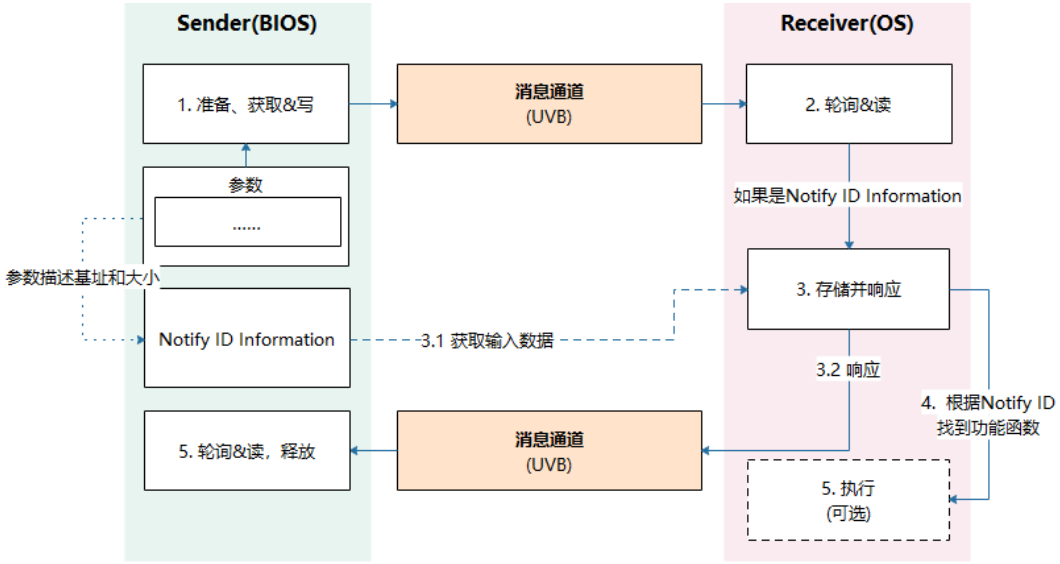

图 B.2 采用 UVB 上报的典型过程

采用 UVB 方式上报步骤描述如下：

步骤 1：Sender（BIOS）准备好需上报的信息，并占用一个 UVB 窗口，然后将 Notify ID 和参数信息按 UVB 窗口格式要求写入窗口；

步骤 2：Receiver（OS）轮询 UVB 通道，转存所有发给自己的 Notify ID Information；

步骤 3：消息转存程序读取参数信息指定的消息，保存到 Receiver 本地用于接收 Notify ID Information 的空间，并返回响应；

步骤 4：根据 Notify ID 找到指定处理程序，将消息地址作为入参输入。

步骤 5：发送侧释放虚拟总线窗口占用；接收侧处理消息（可选，由实现决定）。

b) 中断（Notify Interrupt）方式上报

采用中断上报的典型过程见图 B.3。

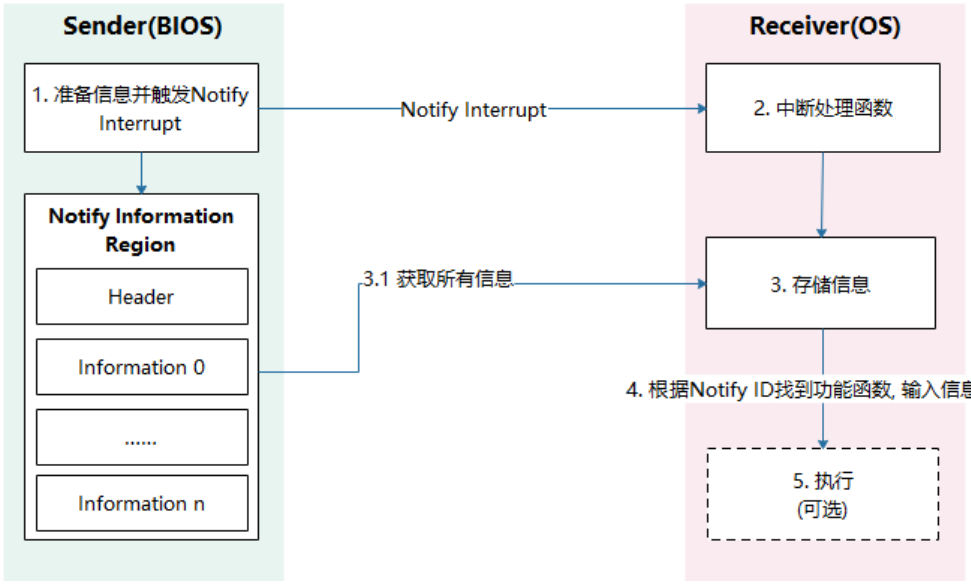

图 B.3 采用中断上报的典型过程

采用中断上报方式的步骤描述如下：

步骤 1：Sender（BIOS）准备好需上报的 Notify ID Information 之后，Notify ID Information 被写到事先与 Receiver（OS）约定的地址（详见 Notify 信息表），触发 Notify Interrupt，通知 Receiver（OS）；

步骤 2：Receiver（OS）中断处理程序执行消息转存功能；

步骤 3：消息转存功能读取所有有效的 Notify ID Information，并保存到 Receiver 本地空间；

步骤 4、步骤 5：接收侧处理同 UVB 方式上报。

注：此处 3.1 为实线，表示此地址未经过中间层转换，而是双方约定共同可访问的地址。

B.3 BIOS 与 BMC 间接口应用

B.3.1 概述

运行在不同系统中的BIOS与BMC，应通过彼此连接的物理通道传递数据，采用其他协议交互（如IPMI），报文数据格式通常采用Type1。

B.3.2 Call ID Service 应用场景

BIOS 与 BMC 间 Call ID Service 接口是双向的，可以彼此相互调用，典型过程见图 B.4。

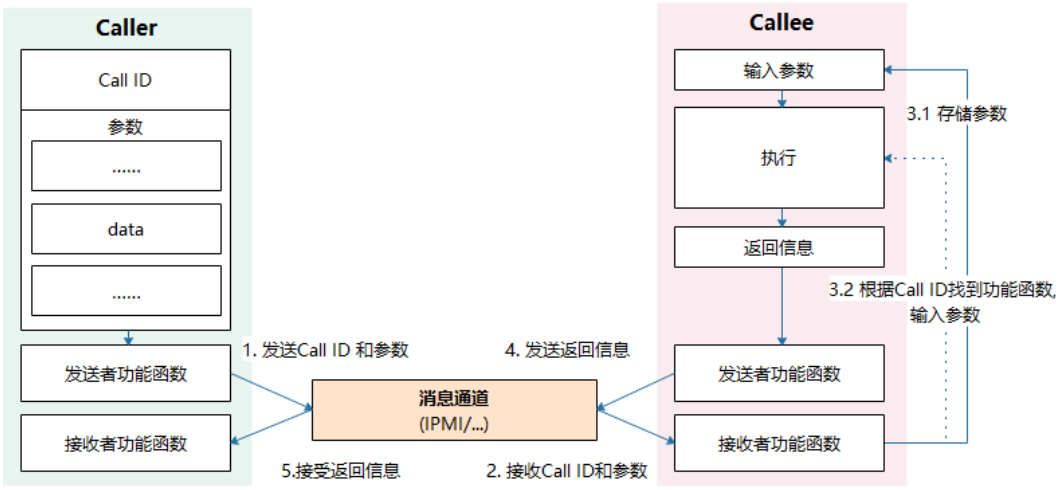

图 B.4 BIOS 与 BMC 间 Call ID Service 调用过程

步骤 1：Caller 准备好 Call ID 和参数，通过发送功能向消息通道发送，此处消息通道指 IPMI 等其他协议通道，此处发送的参数并不是参数地址，而是完整的入参数据包；

步骤 2：Callee 接收 Caller 发送的 Call ID 和入参数据；

步骤 3：（1）Callee 将接收到的入参数据保存到内部存储空间；（2）Callee 根据 Call ID 定位到具体功能，并将保存入参的存储空间地址作为入参输入；

步骤 4：Callee 将返回信息通过消息通道发送给 Caller；

步骤5：Caller接收Callee发送的返回信息，填入参数指定的地址。

B.3.3 Notify ID Information 应用场景

BIOS 与 BMC 间 Notify ID Information 接口通常是单向的，由 BIOS 向 BMC 传递，典型的过程见图 B.5。

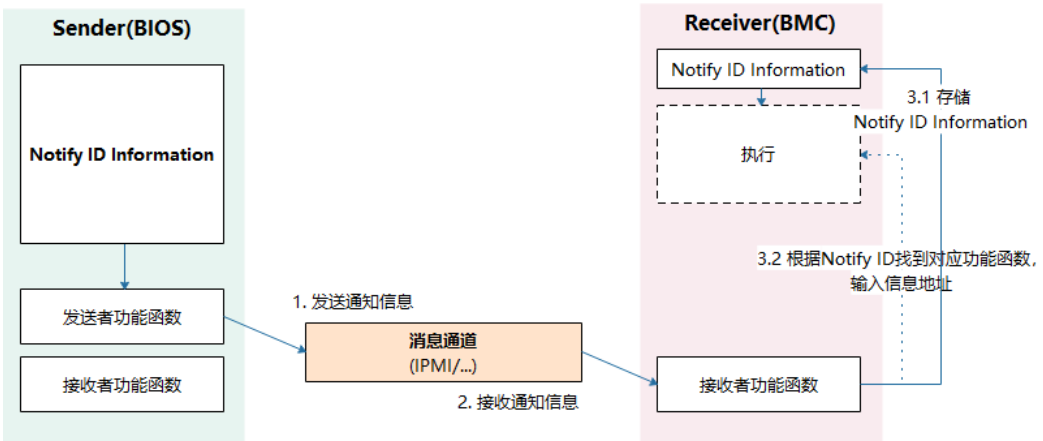

图 B.5 BIOS 与 BMC 间 Notify ID Information 调用过程

步骤 1：Sender（BIOS）准备好需上报的信息，通过消息通道发送；

步骤 2：Receiver（BMC）接收 Sender（BIOS）发送的信息；

步骤 3：（1）Receiver（BMC）将收到的信息转存到本地存储空间；（2）Receiver（BMC）执行消息处理程序，并将存储到本地的消息地址作为入参输入。

注：此方式传递 Notify ID Information 无需要响应；采用“其他（通信）协议”时是否需要响应由被选用的通信协议决定。

B.4 BIOS 与外设间接口应用

B.4.1 概述

BIOS 与外设间数据传递强依赖外设与主 CPU 的物理连接通道。

对于支持 BIOS 与外设间的空间共享的通道，应采用 BIOS 与 OS 间数据传递方式的方式进行接口应用。对于无法支持空间共享的通道，应采用符合 BIOS 与 BMC 间数据传递的方式进行接口应用。

B.4.2 Call ID Service 应用场景

BIOS 与外设间 Call ID Service 接口是单向的，只允许 BIOS 调用外设。此时需要外设给 BIOS 提供 Call ID Service 信息表。

B.4.3 Notify ID Information 应用场景

BIOS与外设间Notify ID Information接口是单向的，只能外设向BIOS发送。