Connect系统

---总体设计

1.0

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **修订人** | **修订内容** |
| 1.0 | 2022/4/27 | 曾毅 | 初稿 |
|  |  |  |  |

目 录

[Connect系统 1](#_Toc27302)

[---总体设计 1](#_Toc3755)

[1. 目的 6](#_Toc25759)

[1.1 范围 6](#_Toc20778)

[1.1.1 业务功能 6](#_Toc25344)

[1.1.2 设备业务 11](#_Toc32527)

[1.1.3 管理业务 11](#_Toc28256)

[1.1.4 系统维护业务 12](#_Toc27999)

[1.1.5 性能容量 12](#_Toc827)

[1.1.6 非功能需求 13](#_Toc23566)

[1.1.7 产品形态 13](#_Toc18830)

[2. 功能概况 15](#_Toc918)

[2.1 系统概况 15](#_Toc20118)

[2.2 硬件特征列表 15](#_Toc16449)

[2.2.1 描述 15](#_Toc31084)

[2.2.2 系统结构框图 15](#_Toc20692)

[2.2.3 外部接口 15](#_Toc31118)

[2.3 软件/固件特性列表 15](#_Toc2939)

[2.3.1 描述 15](#_Toc9875)

[2.3.2 外部接口 15](#_Toc17187)

[2.4 本项目未包含的特性 16](#_Toc17050)

[2.5 软件影响评估 16](#_Toc10412)

[2.6 可测试性考虑 16](#_Toc23873)

[2.7 制造考虑 16](#_Toc3731)

[2.8 网管考虑 16](#_Toc17177)

[2.9 专利考虑 16](#_Toc27397)

[2.10 商标(赝品)保护考虑 16](#_Toc1524)

[3. 外部规格说明 18](#_Toc4991)

[3.1 概述 18](#_Toc19593)

[3.2 功能需求 18](#_Toc18013)

[3.3 性能需求 18](#_Toc19512)

[3.4 易用性需求 18](#_Toc31249)

[3.5 外部接口需求 18](#_Toc9721)

[3.6 架构基线考虑 18](#_Toc26324)

[3.7 约束性需求 18](#_Toc26557)

[3.7.1 硬性约束需求 18](#_Toc9992)

[3.7.2 产品演化规划，PEP 19](#_Toc15809)

[3.8 品质需求 19](#_Toc27574)

[3.8.1 安全需求 19](#_Toc31660)

[3.9 可靠性/可用性需求 19](#_Toc28089)

[4. 内部规格说明 20](#_Toc9072)

[4.1 概述 20](#_Toc18850)

[4.1.1 数据管理 21](#_Toc59)

[4.1.2 组件化 22](#_Toc9044)

[4.2 业务模型 24](#_Toc24070)

[4.2.1 业务层次 24](#_Toc6123)

[4.2.2 簇群业务 25](#_Toc27264)

[4.2.3 网络业务 26](#_Toc30569)

[4.2.4 设备业务 26](#_Toc29284)

[4.2.5 系统业务 26](#_Toc19774)

[4.2.6 软件数据流 28](#_Toc1666)

[4.3 功能模型 29](#_Toc13739)

[4.3.1 功能层次模型 29](#_Toc11776)

[4.3.2 系统服务 31](#_Toc12165)

[4.3.3 通用功能 32](#_Toc14230)

[4.4 业务关系 32](#_Toc21747)

[4.4.1 业务依赖 33](#_Toc2525)

[4.4.2 系统业务 vs 系统服务 34](#_Toc31500)

[4.5 初始化 34](#_Toc4841)

[4.5.1 进程初始化 34](#_Toc30600)

[4.5.2 设备初始化 35](#_Toc14779)

[4.5.3 簇群初始化 35](#_Toc1250)

[4.6 代码组织结构 35](#_Toc2137)

[4.6.1 代码组织 35](#_Toc16541)

[4.6.2 组件包 38](#_Toc15243)

[4.7 编译和打包 42](#_Toc23683)

[4.7.1 传统编译打包 42](#_Toc3465)

[4.7.2 组件化编译打包 42](#_Toc8987)

[4.7.3 编译打包工程版本控制 42](#_Toc31525)

[4.8 主要组件 43](#_Toc12467)

[4.9 软件内存估计 43](#_Toc32564)

[4.10 性能 43](#_Toc3489)

[5. 问题，风险和依赖 44](#_Toc24125)

[5.1 平台需求 44](#_Toc31661)

[5.2 相关项目 44](#_Toc11457)

[5.3 第三方关联 44](#_Toc6089)

[5.4 唯一来源的组件 44](#_Toc19603)

[5.5 技术需求 44](#_Toc14973)

[5.6 技术风险 44](#_Toc16920)

[6. 附录A-标题 45](#_Toc5294)

[7. 需求可跟踪考虑 46](#_Toc2155)

[8. 参考资料 47](#_Toc13227)

[9. 术语表 48](#_Toc949)

[10. 附件 49](#_Toc11947)

[10.1 评审活动记录 49](#_Toc27453)

# 目的

本文档对Connect系统进行总体设计，Connect系统的目标是成为我司通信设备的通用系统。

## 范围

Connect软件系统。

### 业务功能

基于《交换机产品 需求规格说明书 1.0》分析，交换机支持的业务功能按照网络层次以及业务类型划分分为端口业务，链路层业务，3层业务，隧道业务，策略业务，QoS业务，统计业务，安全业务，集群业务。

#### 端口业务

和端口相关的业务如下，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x61100101 | 端口 | 芯片功能 | 存储转发 |
| R-0x61100102 | 端口 | 芯片功能 | 异常帧检测 |
| R-0x61100100 | 端口 | 芯片功能 | 超长帧 |
| R-0x61101001 | 端口 | 端口基本业务 | 速率配置 |
| R-0x61101002 | 端口 | 端口基本业务 | 流控 |
| R-0x61101101 | 端口 | Jumbo Frame |  |
| R-0x61101201 | 端口 | 风暴抑制 | 广播、组播、DLF单播抑制 |
| R-0x61101202 | 端口 | 风暴抑制 | 基于pps抑制 |
| R-0x61101203 | 端口 | 风暴抑制 | 基于bps抑制 |
| R-0x61101204 | 端口 | 风暴抑制 | 按照带宽比例抑制 |
| R-0x61101301 | 端口 | 端口镜像 | 基于物理口镜像 |
| R-0x61101302 | 端口 | 端口镜像 | 基于VLAN标签镜像 |
| R-0x61101303 | 端口 | 端口镜像 | 基于流镜像 |
| R-0x61101304 | 端口 | 端口镜像 | 支持N:M端口镜像 |
| R-0x61101305 | 端口 | 端口镜像 | RSPAN |
| R-0x61101306 | 端口 | 端口镜像 | ERSPAN |
| R-0x61101401 | 端口 | 端口隔离 |  |
| R-0x61101501 | 端口 | 端口限速 |  |
|  |  |  |  |

#### 链路业务

链路层业务包括链路层控制业务和链路层协议，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x61200101 | 链路 | 端口聚合 | LACP |
| R-0x61200102 | 链路 | 端口聚合 | 静态聚合 |
| R-0x61200201 | 链路 | MAC地址表 | 动态学习 |
| R-0x61200202 | 链路 | MAC地址表 | 老化时间配置 |
| R-0x61200203 | 链路 | MAC地址表 | 静态MAC地址配置 |
| R-0x61200204 | 链路 | MAC地址表 | 支持黑洞MAC地址 |
| R-0x61200205 | 链路 | MAC地址表 | 基于端口限制MAC地址数 |
| R-0x61200206 | 链路 | MAC地址表 | 基于VLAN限制MAC地址数 |
| R-0x61200301 | 链路 | VLAN | 支持基于端口的VLAN |
| R-0x61200302 | 链路 | VLAN | VLAN常用模式 |
| R-0x61200303 | 链路 | VLAN | 支持MAC-base VLAN |
| R-0x61200304 | 链路 | VLAN | 支持protocol-base VLAN |
| R-0x61200305 | 链路 | VLAN | 支持VLAN Mapping（1：1） |
| R-0x61200306 | 链路 | VLAN | 支持VLAN 聚合（N：1） |
| R-0x61200307 | 链路 | VLAN | 支持VLAN 聚合（1：N） |
| R-0x61200308 | 链路 | VLAN | 支持VLAN Pruning |
| R-0x61200309 | 链路 | VLAN | 支持QinQ |
| R-0x61200310 | 链路 | VLAN | 支持灵活QinQ |
| R-0x61200311 | 链路 | VLAN | SVLAN和CVLAN MAC地址学习处理 |
| R-0x61200312 | 链路 | VLAN | 支持GVRP |
| R-0x61200401 | 链路 | 生成树协议 | STP |
| R-0x61200402 | 链路 | 生成树协议 | RSTP |
| R-0x61200403 | 链路 | 生成树协议 | MSTP |
| R-0x61200404 | 链路 | 生成树协议 | 基于端口配置 |
| R-0x61200501 | 链路 | SmartLink |  |
| R-0x61200601 | 链路 | RRPP |  |
| R-0x61200701 | 链路 | ERPS |  |
| R-0x61200801 | 链路 | LLDP |  |
| R-0x61200901 | 链路 | 端口环回检测 |  |
| R-0x61201001 | 链路 | BFD |  |
| R-0x61201101 | 链路 | 跨机箱链路捆绑 | 支持“控制面分离转发面合一” |
| R-0x61201101 | 链路 | 跨机箱链路捆绑 | 1个网管口管理所有设备 |
| R-0x61201103 | 链路 | 跨机箱链路捆绑 | 堆叠分裂探测检测 |
| R-0x61201104 | 链路 | 跨机箱链路捆绑 | 在线软件版本更新（ISSU） |
| R-0x61201105 | 链路 | 跨机箱链路捆绑 | STP支持 |
| R-0x61201106 | 链路 | 跨机箱链路捆绑 | 叠加IPV4、IPV6网关功能 |
| R-0x61201107 | 链路 | 跨机箱链路捆绑 | 二次故障场景恢复 |
|  |  |  |  |

#### IP业务

IP业务分为IPv4业务和IPv6业务，其中有细分为IP基本，IP 3层应用业务和3层路由协议和3层组播，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x61510101 | IP | IPv4基本协议 |  |
| R-0x61510102 | IP | IPv4基本配置 |  |
| R-0x61510103 | IP | ICMP |  |
| R-0x61520101 | IP应用 | DHCP Client |  |
| R-0x61520201 | IP应用 | DHCP Snooping |  |
| R-0x61520202 | IP应用 | DHCP Snooping | DHCP Snooping option82 |
| R-0x61520203 | IP应用 | DHCP Snooping | option82基于端口、子端口 |
| R-0x61520204 | IP应用 | DHCP Snooping | option82 VLAN选择 |
| R-0x61520301 | IP应用 | DHCP Relay |  |
| R-0x61520302 | IP应用 | DHCP Relay | DHCP Relay option82 |
| R-0x61520401 | IP应用 | DHCP Server |  |
| R-0x61520501 | IP应用 | DHCP Proxy |  |
| R-0x61520601 | IP应用 | Ping |  |
| R-0x61520701 | IP应用 | Tracert |  |
| R-0x61530101 | IP路由 | ARP |  |
| R-0x61530201 | IP路由 | 静态路由 |  |
| R-0x61530301 | IP路由 | VRF |  |
| R-0x61530401 | IP路由 | RIPv1/v2 |  |
| R-0x61530501 | IP路由 | OSPFv1/v2 |  |
| R-0x61530601 | IP路由 | IS-IS |  |
| R-0x61530701 | IP路由 | BGP4 |  |
| R-0x61530801 | IP路由 | 等价路由 |  |
| R-0x61530901 | IP路由 | 策略路由 |  |
| R-0x61531001 | IP路由 | VRRP |  |
| R-0x61540101 | IP组播 | IGMP Snooping |  |
| R-0x61540201 | IP组播 | IGMP Proxy |  |
| R-0x61540301 | IP组播 | IGMP Filter | 基于端口 |
| R-0x61540302 | IP组播 | IGMP Filter | 基于VLAN |
| R-0x61540401 | IP组播 | 支持3层组播复制 |  |
| R-0x61540501 | IP组播 | PIM Snooping |  |
| R-0x61540601 | IP组播 | 组播VLAN |  |
| R-0x61540701 | IP组播 | IGMP |  |
| R-0x61540801 | IP组播 | PIM |  |
| R-0x61610101 | IPv6 | IPv6基本协议 |  |
| R-0x61610102 | IPv6 | IPv6地址结构 |  |
| R-0x61610201 | IPv6 | ICMPv6协议 |  |
| R-0x61610301 | IPv6 | PMTU |  |
| R-0x61610401 | IPv6 | 双栈 |  |
| R-0x61620101 | IPv6应用 | DHCPv6 Client |  |
| R-0x61620201 | IPv6应用 | DHCPv6 Snooping |  |
| R-0x61620202 | IPv6应用 | DHCPv6 Snooping | Option 18和Option 37 |
| R-0x61620301 | IPv6应用 | DHCPv6 Relay |  |
| R-0x61620401 | IPv6应用 | IPv6-Ping |  |
| R-0x61620501 | IPv6应用 | IPv6-Tracert |  |
| R-0x61620601 | IPv6应用 | IPv6-Telnet |  |
| R-0x61620701 | IPv6应用 | IPv6-TFTP |  |
| R-0x61620801 | IPv6应用 | Tunnel | 手动配置 |
| R-0x61620802 | IPv6应用 | Tunnel | 6to4 tunnel |
| R-0x61620803 | IPv6应用 | Tunnel | ISATAP tunnel |
| R-0x61620804 | IPv6应用 | Tunnel | IPv6 in IPv6 tunnel |
| R-0x61620805 | IPv6应用 | Tunnel | IPv4 in IPv6 tunnel |
| R-0x61620806 | IPv6应用 | Tunnel | GRE |
| R-0x61630101 | IPv6路由 | ND |  |
| R-0x61630201 | IPv6路由 | RIPng |  |
| R-0x61630301 | IPv6路由 | OSPFv3 |  |
| R-0x61630401 | IPv6路由 | IS-ISv6 |  |
| R-0x61630501 | IPv6路由 | BGP4+ |  |
| R-0x61640101 | IPv6组播 | MLD Snooping |  |
| R-0x61640201 | IPv6组播 | MLD Proxy |  |
| R-0x61640301 | IPv6组播 | MLD v1/v2 |  |
| R-0x61640401 | IPv6组播 | IPv6 PIM |  |
|  |  |  |  |

#### 策略业务

策略业务主要涉及ACL和流分类，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x61700101 | 策略 | 流分类 |  |
| R-0x61700201 | 策略 | 时间段ACL |  |
| R-0x61700301 | 策略 | 双向ACL |  |
| R-0x61700401 | 策略 | VLAN ACL |  |
| R-0x61700501 | 策略 | 流限速 |  |
| R-0x61700502 | 策略 | VLAN限速 |  |
| R-0x61700503 | 策略 | VLAN限速 | 超出行为修改 |
| R-0x61700601 | 策略 | ACL ACTION | 报文重定向 |
|  |  |  |  |

#### QoS业务

QoS业务包括了流标记，流修改，策略调度，队列调度，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x61800101 | 调度 | 优先级重标记 |  |
| R-0x61800201 | 调度 | CAR |  |
| R-0x61800301 | 调度 | 队列映射 | 基于流分类 |
| R-0x61800302 | 调度 | 队列映射 | 基于IPv6 |
| R-0x61800401 | 调度 | 队列带宽保证 |  |
| R-0x61800501 | 调度 | 队列调度算法 |  |
| R-0x61800601 | 调度 | WRED |  |
| R-0x61800701 | 调度 | HOL |  |
|  |  |  |  |

#### 统计业务

统计业务是对流水线中的报文进行统计，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x61900101 | 统计 |  |  |
| R-0x61900201 | 统计 | Telemetry |  |
|  |  |  |  |

#### 安全业务

在网络各层业务中都支持各种安全业务，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x62000101 | 安全 | 802.1X认证 |  |
| R-0x62000201 | 安全 | 集中式MAC地址认证 |  |
| R-0x62000301 | 安全 | RADIUS |  |
| R-0x62000401 | 安全 | SSH 2.0 |  |
| R-0x62000501 | 安全 | 远程登录IP范围设置 |  |
| R-0x62000601 | 安全 | 远程登陆开关 |  |
| R-0x62000701 | 安全 | 端口安全 |  |
| R-0x62000801 | 安全 | PORTAL认证 |  |
| R-0x62000901 | 安全 | EAD |  |
| R-0x62001001 | 安全 | 网络攻击保护 |  |
| R-0x62001101 | 安全 | 动态ARP检测 |  |
| R-0x62001201 | 安全 | BPDU guard |  |
| R-0x62001301 | 安全 | uRPF |  |
| R-0x62001401 | 安全 | IP/Port/MAC绑定 |  |
| R-0x62001501 | 安全 | 协议MD5密文认证 |  |
| R-0x62001601 | 安全 | PKI |  |
|  |  |  |  |

#### 集群业务

集群业务如下，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x62100101 | 集群 | 堆叠 |  |
| R-0x62100201 | 集群 | 虚拟化 |  |
|  |  |  |  |

### 设备业务

设备业务反映设备硬件的运行情况，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x65000101 | 设备业务 | 电源告警 |  |
| R-0x65000201 | 设备业务 | 风扇告警 |  |
| R-0x65000301 | 设备业务 | 温度告警 |  |
| R-0x65000401 | 设备业务 | VCT |  |
| R-0x65000501 | 设备业务 | EEE |  |
| R-0x65000601 | 设备业务 | 端口定时down |  |
|  |  |  |  |

### 管理业务

管理业务主要提供管理服务，而其他业务通过管理服务对用户提供访问接口，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x66000101 | 管理 | 命令行系统 |  |
| R-0x66000102 | 管理 | 命令行系统 | 用户分级管理和口令保护 |
| R-0x66000201 | 管理 | SNMP |  |
| R-0x66000202 | 管理 | SNMP | 基于端口开关 |
| R-0x66000203 | 管理 | SNMP | IPv6支持 |
| R-0x66000204 | 管理 | SNMP | IPv6 MIB库支持 |
| R-0x66000205 | 管理 | SNMP | SNMP安全 |
| R-0x66000206 | 管理 | SNMP | RMON |
| R-0x66000301 | 管理 | WEB |  |
| R-0x66000401 | 管理 | XML网管 |  |
| R-0x66000501 | 管理 | iMC智能管理中心 |  |
| R-0x66000601 | 管理 | OAM |  |
|  |  |  |  |

### 系统维护业务

系统维护业务，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| R-0x67000101 | 维护 | 账号管理 |  |
| R-0x67000102 | 维护 | 账号管理 | 账号密码复杂口令 |
| R-0x67000103 | 维护 | 账号管理 | 口令更换 |
| R-0x67000201 | 维护 | 升级 |  |
| R-0x67000202 | 维护 | 升级 |  |
| R-0x67000301 | 维护 | TELNET |  |
| R-0x67000302 | 维护 | TELNET | 登陆限制 |
| R-0x67000401 | 维护 | 系统日志 |  |
| R-0x67000402 | 维护 | 系统日志 | 日志上传 |
| R-0x67000501 | 维护 | 告警 |  |
| R-0x67000502 | 维护 | 告警 | 分级告警 |
| R-0x67000503 | 维护 | 告警 | 移动告警标准化梳理表 |
| R-0x67000601 | 维护 | 调试信息 |  |
| R-0x67000701 | 维护 | NTP |  |
| R-0x67000702 | 维护 | NTP | NTP认证 |
| R-0x67000801 | 维护 | 重启 |  |
| R-0x67000901 | 维护 | 复位 |  |
| R-0x67001001 | 维护 | 配置备份与恢复 |  |
|  |  |  |  |

### 性能容量

功能规格中涉及性能容量规格如下，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| *TBD* | *TBD* | *TBD* | *TBD* |
|  |  |  |  |
|  |  |  |  |

### 非功能需求

**异常场景需求**

设备支持双分区备份，避免升级过程中出现异常导致设备无法启动。

**可修改，扩展性需求**

我们需要一套代码适配所有的产品种类，由于不同种类的产品功能有一些差异，而且针对不同的客户有一些单独的需求，所以需要让业务能够动态拆卸，针对不同的产品和不同的客户安装不同的功能。

**生产制造需求**

对于产品设备如要满足生产制造需求，要从几个方面考虑，

1. 能够辅助硬件测试；
2. 通过SDV测试（EMI测试）；
3. 通过SIT测试；
4. 提供生产需要的测试软件版本；

**易用性需求**

易用性主要涉及产品外观和接口标识方面，在功能层面应该考虑对用户提供的命令，SNMP MIB节点和WEB页面遵循行业一般使用原则，简单清晰。

**安全性场景需求**

|  |  |
| --- | --- |
| **安全性** | **描述** |
| 1 标识安全 | 产品的外观安全标识由硬件团队负责 |
| 2 冗余与备份恢复安全 | 固件分区冗余，配置文件冗余 |
| 3 漏洞及恶意程序管理 | 漏洞检查 |
| 4 软件启动及更新安全 | 系统启动暂不检查固件安全性，只做合法性检查，更新固件检查安全性 |
| 5 身份标识与鉴别 | 提供账号密码登陆 |
| 6 访问控制安全 | 提供账号密码登陆 |
| 7 日志审计安全 | 关键业务日志跟踪 |
| 8 通信安全 | 提供防火墙 |
| 9 数据安全 | 升级文件提供数据校验 |

**可测试性需求**

数据通道可测试，出现异常可以定位故障点。

业务流可测试，出现异常可以定位异常进程。

### 产品形态

主要的外部指标如下，

*TBD*

产品外观如下，

*TBD*

# 功能概况

## 系统概况

*TBD*

## 硬件特征列表

### 描述

*TBD，这里应当描述典型的硬件模型*

### 系统结构框图

*TBD，这里应当描述典型的硬件原理框图*

### 外部接口

*TBD，硬件提供设备接口*

## 软件/固件特性列表

### 描述

各种业务提供各自服务，本章节只对业务影响系统设计的部分进行分析，详细的业务在后续设计中逐级拆解。

软件业务参考1.1.1章节。

软件特征就是对需求的进一步分析，将需求整理为研发可以执行的项目，经过整理后的需求可以被研发更好的评估和分析，由于文档作者对需求都比较了解，这里为了节约时间就不做详细分析，后续由研发人员在下一级文档中描述。

### 外部接口

外部接口有命令行，SNMP和WEB，命令行外部接口通过命令行手册描述，SNMP通过MIB节点描述，WEB通过WEB使用手册描述。

## 本项目未包含的特性

由于项目进度的原因以及目标用户的需求，以下功能目前未被包括，

|  |  |  |  |
| --- | --- | --- | --- |
| **SPEC-ID** | **一级SPEC** | **二级SPEC** | **三级SPEC** |
| *TBD* | *TBD* | *TBD* | *TBD* |
|  |  |  |  |
|  |  |  |  |

另外不支持的功能特性参考《交换机产品 需求规格》有详细标注。

## 软件影响评估

暂无。

## 可测试性考虑

对芯片流水线各个阶段的统计信息进行维护，对芯片中的错误寄存器和调试寄存器进行维护。

对SDK的调试命令进行维护。

对管理报文和控制报文进行监控。

## 制造考虑

为了方便制造可提供调试配置命令快速配置产测环境。

## 网管考虑

网管考虑使用第三方平台，考虑使用MIB节点对接。

## 专利考虑

暂无。

## 商标(赝品)保护考虑

暂不涉及。

# 外部规格说明

## 概述

功能的详细功能点和性能容量规格应该在《交换机产品 需求规格说明书》以及相关文件中进行详细阐述并且作为后续研发验收的标准，而本文档的本章节应该对项目中的所有功能的使用和验收进行定义和约束。

## 功能需求

功能需求分析是软件系统设计的重要导入部分，但是考虑到详细分析花费时间较多，而文档作者对功能已经比较了解，故本章节暂略。

## 性能需求

性能需求参考章节1.1.2。

## 易用性需求

*TBD*

## 外部接口需求

*TBD*

## 架构基线考虑

全新的Connect系统无基线参考。

## 约束性需求

### 硬性约束需求

*硬件提供*

### 产品演化规划，PEP

支持不同芯片的接入汇聚方案，端口数量可以覆盖8，16，24，48口（光or电）。

## 品质需求

### 安全需求

|  |  |  |
| --- | --- | --- |
| **法律法规** | **适用产品** | **验收方式** |
| 《网络安全法》（必须） | 通用 |  |
| 《网络产品和服务安全审查办法（试行）》 （必须） | 通用 |  |
| 《GB T 35273-2017个人信息安全规范》 （必须） | MCP,MACC,SMP,SAM等 |  |
| 《网络关键设备或网络安全专用产品目录》(第一批)  （必须） | 通用 |  |
| 《关键信息基础设施安全保护条例（征求意见稿）》 | 通用 |  |
| 《未成年人网络保护条例》 | 通用 |  |
| 《工业控制系统信息安全防护指南》 | 通用 |  |
| 《关键信息基础设施》目录 | 通用 |  |
| 《国家网络安全事件应急预案》 | 通用 |  |

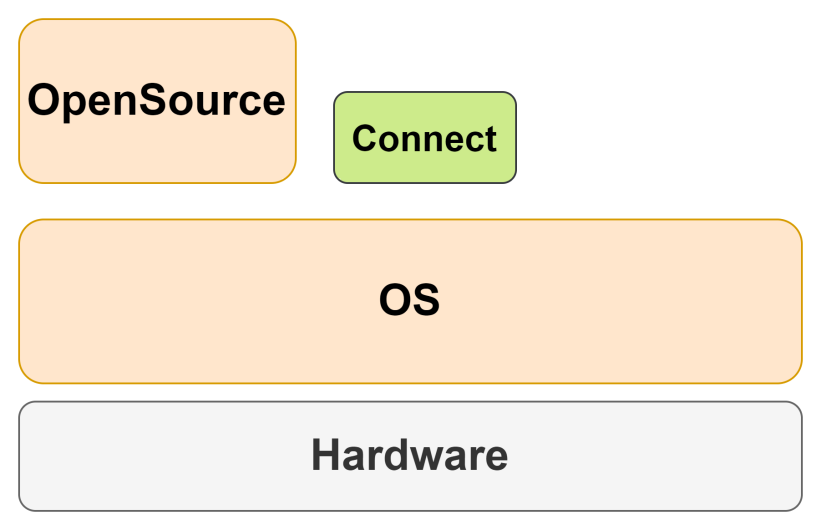
## 可靠性/可用性需求

设备硬件的可靠性和可用性依赖硬件设计保证，通过MTBF测试进行验证。

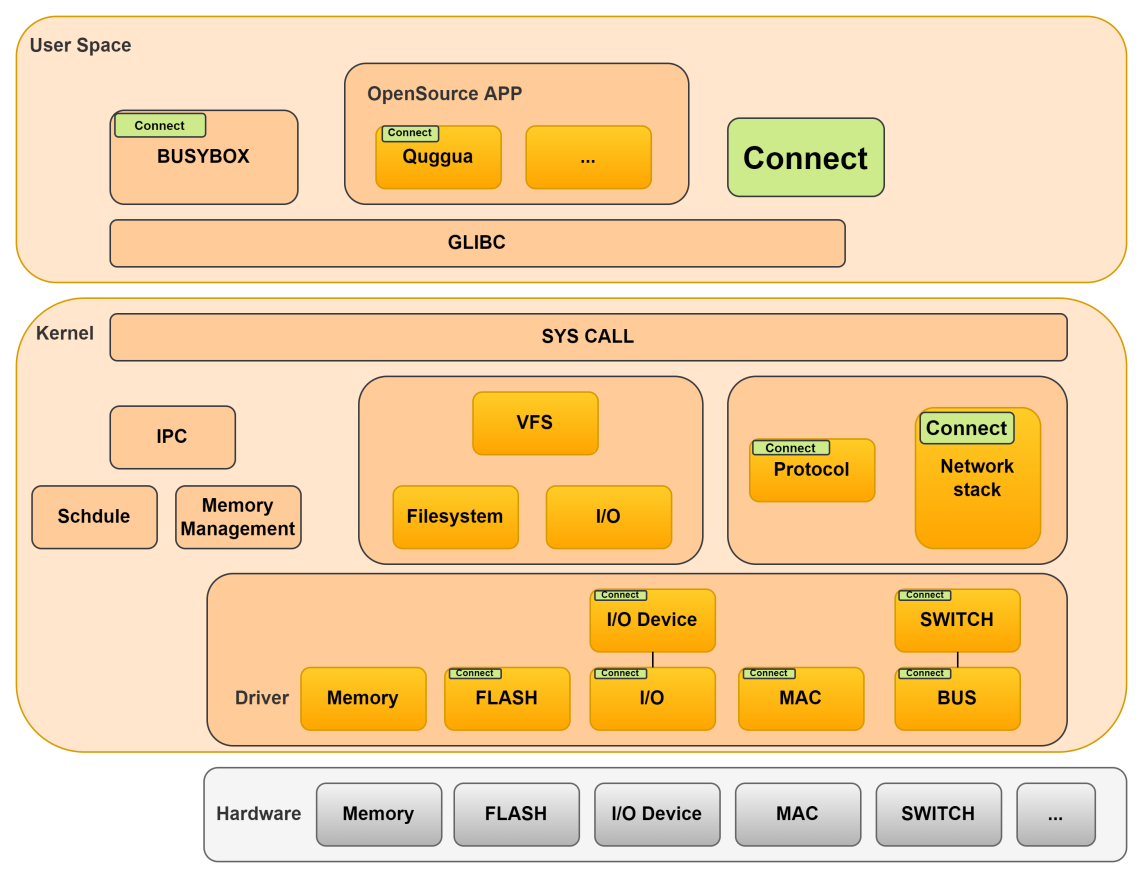
# 内部规格说明

## 概述

我司软件系统由Linux操作系统，开源组件以及我司自主研发的Connect通信软件系统组成，如下图所示，



Connect系统的主要组件由我司自主研发，同时为了更好和Linux系统融合以及融合开源组件，Connect系统针对Linux系统和开源组件有一些适配代码。如下所示Connect系统在Linux系统中的关系，



Connect系统总体上要考虑数据管理和组件化以满足系统未来的稳定性和扩展性需求。

### 数据管理

系统运行过程中会产生许多数据，包括设备硬件自身的能力，产品和项目定义的功能规格，用户对设备的配置，业务模块初始化配置以及业务模块运行中产生的状态数据和动态表项等，这些数据由各个业务模块自行维护，但是为了使Connect系统的数据更加聚合同时满足簇群业务需求，我们把数据进行结构化管理。

#### 数据结构化

数据结构化是簇群的基础，在系统中设备能力，功能规格，配置数据，状态数据和动态表项需要进行数据结构化。

数据结构化在逻辑上是树形结构，

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **簇群ID** |  |  | **静态库** | **运行数据** | **数据库** |
| 能力配置 | 簇群能力配置 | 簇群业务A能力 | ✔ |  | ✔ |
| 簇群业务B能力 | ✔ |  | ✔ |
| ... | ✔ |  | ✔ |
| 设备能力配置 | 设备业务A能力 | ✔ |  | ✔ |
| 设备业务B能力 | ✔ |  | ✔ |
| ... | ✔ |  | ✔ |
| 业务配置 | 簇群业务配置 | 簇群业务A配置 |  | ✔ | ✔ |
| 簇群业务B配置 |  | ✔ | ✔ |
| ... |  | ✔ | ✔ |
| 设备业务配置 | 设备业务A配置 |  | ✔ | ✔ |
| 设备业务B配置 |  | ✔ | ✔ |
| ... |  | ✔ | ✔ |
| 业务状态 | 簇群业务状态 | 簇群业务A状态 |  | ✔ | ✔（可选） |
| 簇群业务B状态 |  | ✔ | ✔（可选） |
| ... |  | ✔ | ✔（可选） |
| 设备业务状态 | 设备业务A状态 |  | ✔ | ✔（可选） |
| 设备业务B状态 |  | ✔ | ✔（可选） |
| ... |  | ✔ | ✔（可选） |
| 业务动态表项 | 簇群业务表 | 簇群业务A表项 |  | ✔ | ✔ |
| 簇群业务B表项 |  | ✔ | ✔ |
| ... |  | ✔ | ✔ |
| 设备业务表 | 设备业务A表项 |  | ✔ | ✔ |
| 设备业务B表项 |  | ✔ | ✔ |
| ... |  | ✔ | ✔ |
|  |  |  |  |  |  |

#### 能力数据和功能规格

能力数据和功能规格都可以分为静态配置和动态配置，静态配置在编译阶段根据项目和产品就已经确定，这些配置信息可以直接通过库文件获取，而动态配置可以在运行态时动态变化，动态能力配置将通过数据库加缓存的方式获取。

### 组件化

我们基于Linux平台，如果要提高系统的稳定性降低新业务加入带来的风险，最好的策略就是将业务进行组件化管理，每个组件可以独立升级，组件支持重载，组件之间耦合低相互影响小，组件由可控数量的进程和库组成。

#### 组件划分

组件按照层次和业务类型进行合理划分（层次结构参考4.2和4.3章节）。

Cluster Management Layer，Cluster Control Layer，Device Control Layer，Board Control Layer，System Service Layer和System Common Layer的组件彼此分离。

对于业务按照网络业务，设备业务和系统业务进行组件包划分。网络业务按照网络层次和业务特性做进一步组件划分，比如端口组件，链路层控制组件，生成树组件，IP组件等；设备业务按照设备功能划分组件，如果功能较小则按照重要性归类；每个系统业务都是单独的组件。

对于系统服务，每个服务都是单独组件；对于通用服务，按照功能目的归类组件。

#### 进程管理

组件化落地到系统中就是进程化，对于Connect系统要按照以下约定进行进程划分，

1. 进程不能实现跨层业务，但是考虑到系统性能，设备成本以及系统复杂度，在一个进程中可以同时承载Cluster Control Layer和Device Control Layer业务，但是两层之间的调用必须用API衔接；
2. 管理层服务都是独立进程（命令行，SNMP，WEB等），而各个业务作为其中的适配代码放入其中，需要注意的是所有的业务代码都要支持进程重载；
3. 在控制层中网络业务，设备业务和系统业务进程独立；
4. 网络业务中协议和纯控制业务进程独立；
5. 如果业务较小一般情况不要使用单独进程维护，各种业务可以在满足前面几点约束的前提下按照业务的重要性进行聚合。

#### 进程间通信

进程间通信是组件化重要的组成部分，组件之间涉及跨进程通信，跨板卡通信和跨设备通信。Linux操作系统提供了许多IPC通信方式比如Unix Socket，Pipe，TCP/IP Socket等，对于Connect系统来说这仅仅是通信层，根据业务之间的业务依赖关系不同进程间通信要支持不同的服务模型。

##### **异步依赖 & 同步依赖**

首先业务之间的依赖关系大体上可以分为异步依赖和同步依赖，异步依赖又可以分为异步消息依赖和异步回复依赖，异步消息依赖又分为可以丢失消息依赖和不可丢失消息依赖。

对于异步消息可以丢失的情况大体上是一些持续上报的信息，这些信息偶尔丢失一两次不影响业务运行；

对于异步消息不可丢失不可乱序的情况，大体上是低耦合业务之间的事件通知，收到消息的业务会做许多业务处理；

对于异步消息等待回复的情况，这种一般都是同步业务需求，但是业务处理花费时间太久所以改为异步模式，这种情况需要注意的是异步处理在多个业务是同时进行的，如果考虑不充分可能出现死锁或者资源抢占等问题；

同步依赖就是会等待对方回复，考虑到如果在等待过程中对方异常退出等问题导致的严重后果，同步依赖应该支持超时机制。

##### **异步通信 vs 同步通信**

异步通信效率较高，但是异步通信有两个基本问题就是丢包和乱序，对于丢包问题一般要分情况看有的业务可以容忍丢包但绝大多数业务都不能丢包，而业务一般都不能接受乱序问题。

同步通信效率较低，但是业务行为顺序可控。

异步通信在面对1对多通信时，丢包和乱序问题会更加严重，而带来的重发重排的设计复杂度会更高，同步通信在进行1对多通信时，效率会更低因为只有当所有回复都响应后才会认为此次通信结束。

异步通信和同步通信各自的优缺点很明显，根据业务依赖的不同考虑使用不同的通信是理性的做法。

不管业务之间是异步依赖还是同步依赖，采用异步通信还是同步通信其通信层都是IPC也就是异步通信，要区分不同逻辑层次上的异步同步。

##### **IPC SWITCH**

考虑到异步通信和同步通信都需要支持1对1，1对多的通信方式，另外系统中的进程较多，如果每个进程都要和其他进程进行通信那么需要建立N\*N的IPC连接，为了减少IPC连接带来的设计复杂度和系统资源开销，系统考虑提供IPC SWITCH服务，即所有进程都和IPC SWITCH建立通信并注册地址，当一个进程要和其他进程通信时它将把信息发送到IPC SWITCH，由IPC SWITCH负责选路。

IPC SWITCH支持单发和组播发送。

接下来的章节将从业务模型，功能模型，业务关系，初始化，编译和打包等多个角度多Connect系统进行分析。

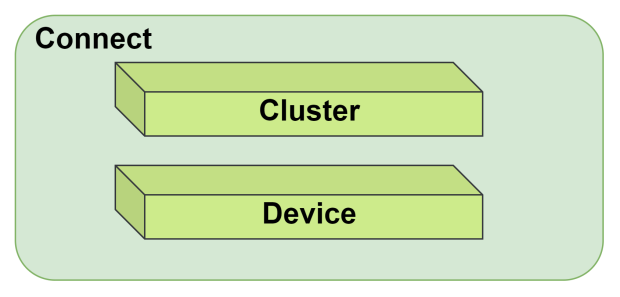
## 业务模型

业务模型从业务视图的角度对系统进行说明，

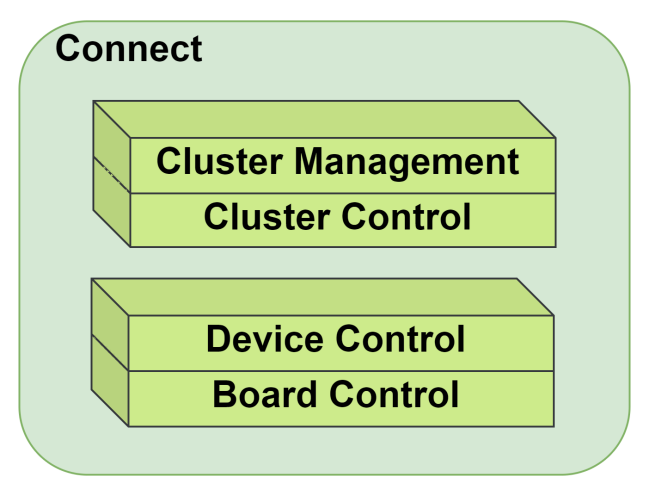
### 业务层次

从业务的角度Connect系统分为两层：Cluster层和Device层。

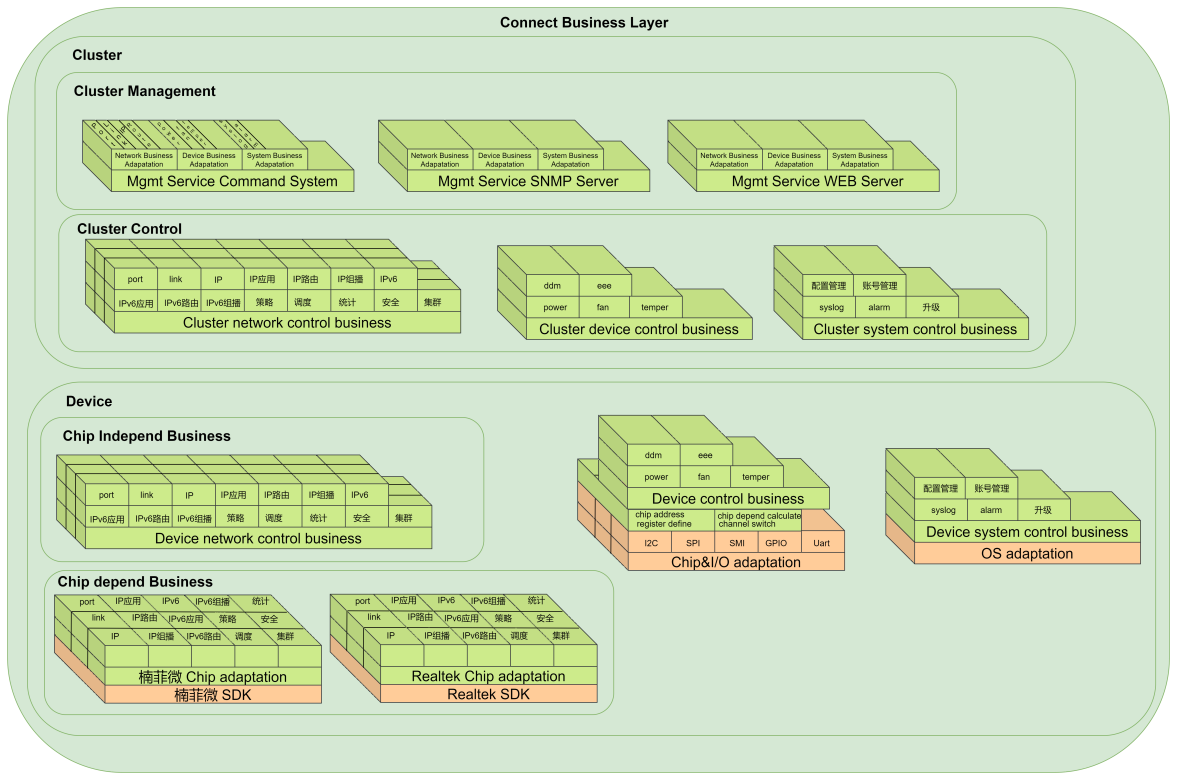
Cluster层呈现整个簇群设备的管理接口并运行簇群级的业务，Device层运行设备级的业务。



Cluster层从业务的角度可以拆分为两层即管理层和控制层，管理层将对用户提供管理接口，而控制层运行基于Cluster层级的控制业务；Device层同样从业务的角度可以拆分为两层即设备层和板卡层，设备层运行基于整台物理设备的控制业务，而板卡层运行单卡控制业务。根据设备复杂度不同有一些业务可以在Cluster控制层和Device控制层之间移动。



详细的业务层次图如下，下图只是例子不完全包括所有业务，可以看到Cluster层业务设备无关性较强，而Device层业务和设备硬件有较大关联，如果Device层进一步拆分为Device层和Board层那么Device层主要运行基于所有板卡的业务以及配置下发，而Board层将承载硬件适配业务。



### 簇群业务

系统要支持簇群业务，簇群业务就是把一些设备通过指定端口通过内部协议将它们协商成为逻辑上的单台设备并对用户提供统一管理层的业务。

我们通过将Connect系统抽象为Cluster层和Device层可以很好的实现簇群业务，单台设备启动以后通过开启Cluster控制层的簇群功能将自动协商或者加入某个簇群。

簇群的协议以及管理报文流由相关的下一级设计进行分解。

#### 簇群虚拟化

簇群虚拟化分为两个部分，一个是设备虚拟化，另外一个是簇群虚拟化。

实现虚拟化可能通过多种方式，

第一种是将数据和逻辑分离，这种方式是在编码阶段完成，对系统的资源要求最少，但是开发难度较高，出错率也高，一旦某个虚拟化逻辑异常导致所有虚拟化设备均异常；

第二种也是将数据和逻辑分离，同时基于进程进行隔离，对每个虚拟化设备提供独立的一组进程，这样的好处是开发难度降低，如果进程异常虚拟设备之间不会相互影响，但是这种做法会导致系统内的业务进程较多通信服务变得复杂，服务业务设计难度颇高，比如进程管理，配置管理，系统日志，系统告警要基于虚拟设备进行信息隔离；

第三种是借助容器进行系统空间隔离，通过容器在系统内运行多套独立逻辑，但是在硬件层或者驱动层尤其是一些共享接口和共享资源无法完全分离，还是要做复用处理。这种方式对编码要求最低，但是对系统资源要求最高。

簇群虚拟化的一个基本要求是数据结构化，即虚拟化设备的所有数据包括配置，信息以及中间运行数据都是基于虚拟化设备域有效。

我们使用第二种和第三种方式结合来实现簇群虚拟化，

第一步将能力数据，配置数据和状态数据等数据和业务代码分离，这样业务代码中只存在一些临时数据；

第二步将业务代码基于簇群进程化，但是对设备业务，系统业务和系统服务仍然保持原样，设备业务虚拟化没有太大意义除非用户有特殊要求，而设备级的系统业务没有必要进行虚拟化，比如设备升级，另外一些系统业务，比如账号管理，配置保存，系统复位等业务要基于数据进行虚拟化设计，由于业务本身就是服务型的，所以不考虑多进程化，因为这些业务进程本身就是可重载的，对于系统服务和系统业务一样，基于设备的系统服务不考虑虚拟化，其他的服务则基于数据进行设计；

第三步增加容器，这时容器内的业务基于能力文件启动即可。对于硬件资源以及芯片相关的驱动层应该注意簇群虚拟化能力域的处理，比如泛洪到VLAN所有成员应该限制在某个虚拟簇群中，而虚拟簇群之间的隔离有独立的模块进行处理。

### 网络业务

#### 配置

业务配置大多由用户发起，通过管理协议报文传递给Cluster服务，服务解析报文调用业务适配代码，适配代码通过IPC访问控制层实现业务。

#### 信息获取

信息获取的方式比较灵活，如果是配置信息可以直接通过Cluster数据库获取，而状态和表项则根据具体的业务可以从数据库或者通过IPC从控制层获取。

#### 协议

网络业务除了配置和信息获取还包括网络协议，网络协议报文通过控制协议流通路到业务模块进行处理。

### 设备业务

设备业务和网络业务的配置和信息获取流程一样。

### 系统业务

系统业务的配置以及内容获取流程上基本和网络业务的流程相同，系统业务最大的区别在于由于业务是在系统平面，一般来说其他业务都会和系统平面的业务存在业务联系。

下面的子章节将描述典型的系统业务，

#### 复位

复位功能是将设备恢复到出厂状态，对Connect系统来说，其中的所有业务都要恢复，这时复位业务和其他业务是一种1对多的请求和响应关系，各个业务在进行复位时比较耗时，所以复位功能对其他业务复位时应该采用一种异步响应方式。这种异步响应机制不需要复位功能实现，而应该有系统服务层提供相应业务。

对于各个业务来说，由于复位是异步进行，在进行异步复位处理时应该注意最好不要和普通的配置删除业务流程复用，因为异步复位时可能会忽略和其他业务之间的访问和检查直接清楚自己的数据和状态。

复位也可以采用同步的机制，即各个业务向复位业务注册复位顺序，复位时只有上一个业务响应复位结果后才通知下一个业务进行复位。同步复位顺序可控，但是效率太低，当设备配置比较多的时候如果依次按业务复位，业务复位时操作硬件会产生大量的I/O等待造成复位时间非常长。

异步复位最大的问题是复位顺序不可控，可能造成删除依赖导致一些配置无法清楚但是如果业务通过单独处理复位流程即可规避这一问题。

#### 配置保存

配置保存业务是将所有业务的配置保存，所以在业务上是1对多的请求和响应关系，相比复位配置保存更麻烦，复位只是业务自身清楚配置，而配置保存需要业务将配置按照一定方式格式化并交由配置保存业务处理。

从业务上看，配置的顺序有强依赖要求。

配置保存有两种方式，

一种是Cluster控制层的配置数据库直接存储到FLASH并且加密，用户可以将数据导出到其他设备再导入，这种方式将配置的顺序问题交给配置加载业务解决；

另一种是Cluster控制层生成命令行配置文件，配置文件中命令的顺序问题可以通过多种方式解决，第一，可以通过各个业务携带保存顺序解决，第二，可以由配置保存业务直接读取数据库生成配置文件。

#### 配置加载

根据加载的方式不同，配置加载业务和其他业务之间的访问方式也不同。如果配置加载业务采用数据库加载，当数据库加载完毕以后将按照一定的顺序通知其他业务加载数据库配置，如果配置加载业务采用配置文件加载，则通过启动命令行引擎加载配置文件即可。

#### 升级业务

升级业务和前面三个系统业务不同，属于比较内敛的系统业务。

**簇群升级模型**

簇群升级主要是对打包文件的定义以及簇群设备之间文件传输的设计，对单台设备来说都是设备升级模型。

**设备升级模型**

设备升级的主要步骤是下载升级文件，检查升级文件，刷新分区。

**组件升级模型**

组件升级最难的是解决组件之间的版本依赖，要解决这个问题最好引入dpkg等包管理框架。否则组件升级只能针对我们少量耦合度较低的业务。

### 软件数据流

根据业务的特性和需求在软件上应该存在数据平面和协议栈。

#### 软件数据平面模型

对于交换机来说交换芯片承载了数据平台，对于其他设备比如路由器，网关由于链路层不全是以太网，并且也有一些特殊的业务需求，故需要将数据经过软件处理，在这种情况下在软件侧也有一条数据平面，由于数据平面的设计比较复杂，这里只对其进行基本约束，保证后续的兼容性。

软件数据平面的报文收发主要由CPU外围控制器完成，针对控制器的初始化和配置一般是放到内核态中，而针对数据报文的收发则可以通过UIO和MMAP将报文Buffer映射到用户态处理，如果不考虑性能也可以直接使用交换芯片的SDK收发包。

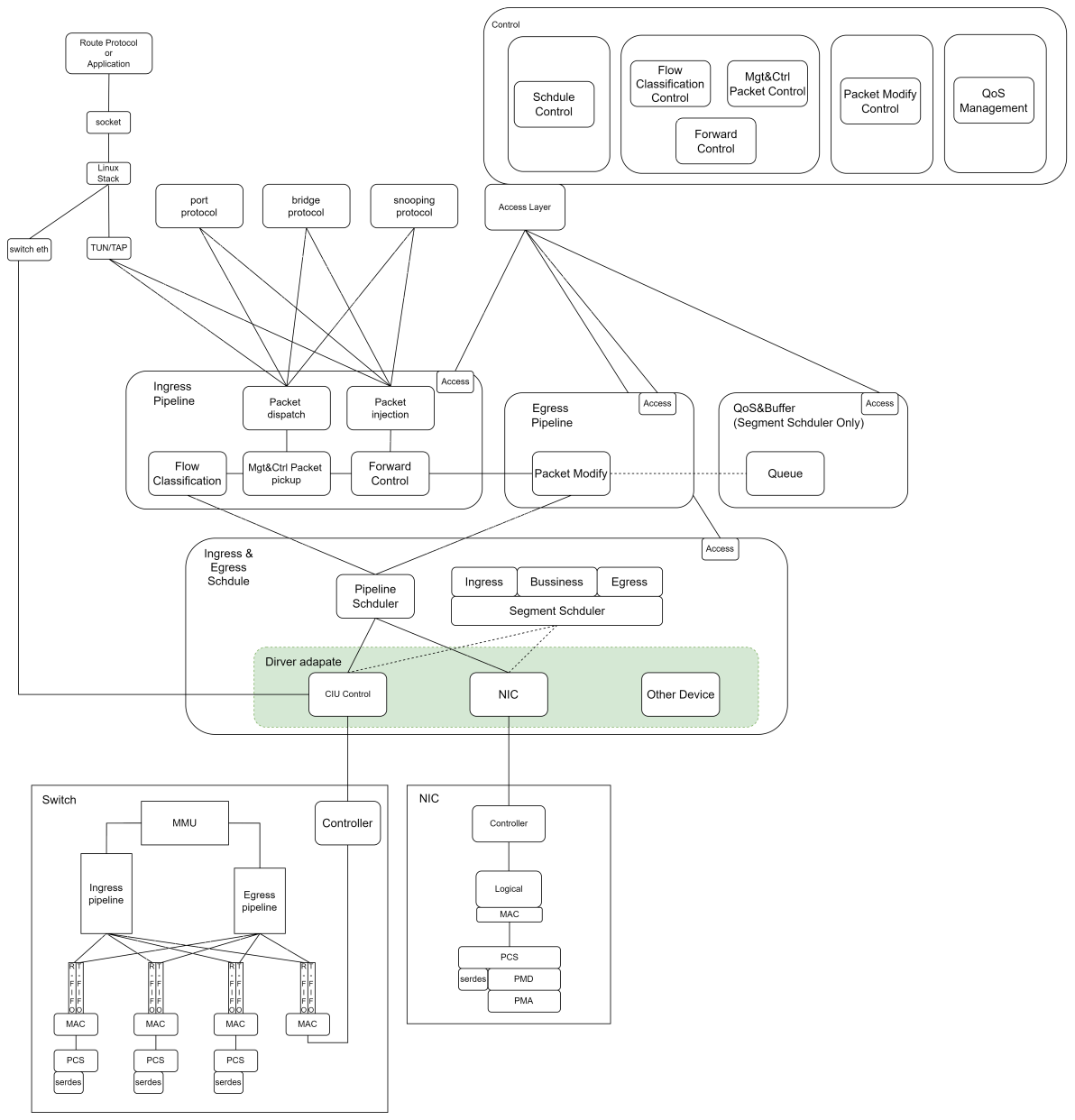
数据报文接收到软件数据平面以后将会按照数据平面流水线处理。

数据报文处理完成以后再通过封包将数据发送。

#### 管理控制平面协议流

到CPU的报文流有多种目的，即使报文是To Me报文根据协议的不同其“Me”的含义也有所不同，对于Bridge协议希望将协议报文发送到端口控制层，基于802.1Q的控制协议则是希望把协议报文发送到端口和VLAN上的控制层，3层应用协议和路由协议则是希望基于VLANIF或者VLANIF+SubIF把协议报文发送到控制层，3层以上的应用协议包括管理会话（Telnet，SSH），管理服务（WEB，SNMP）和安全（AAA）等则是HOST主机需要的内容，3层以上的协议都是将设备抽象为运行协议的主机，而3层以下的协议更倾向将端口或者端口加VLAN作为运行协议的主机单元。

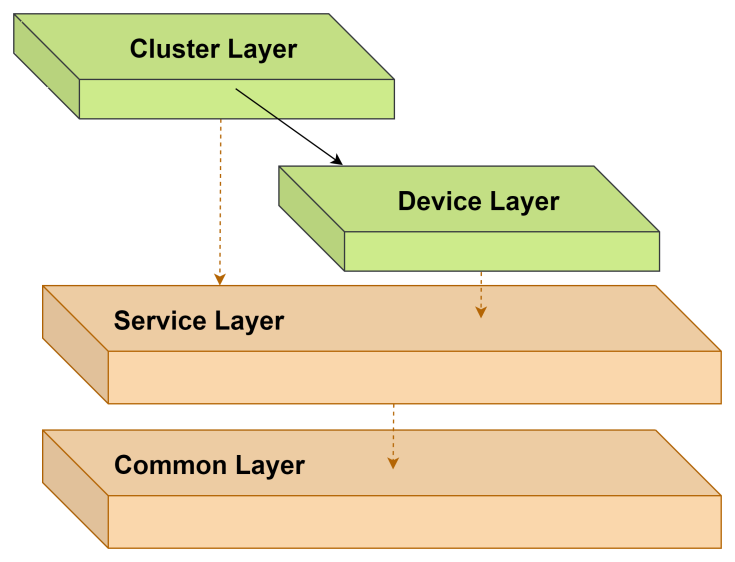
除了To Me的协议报文，还有一类是Snooping协议报文这类报文，更多应该考虑为流水线业务在软件层面上实现。



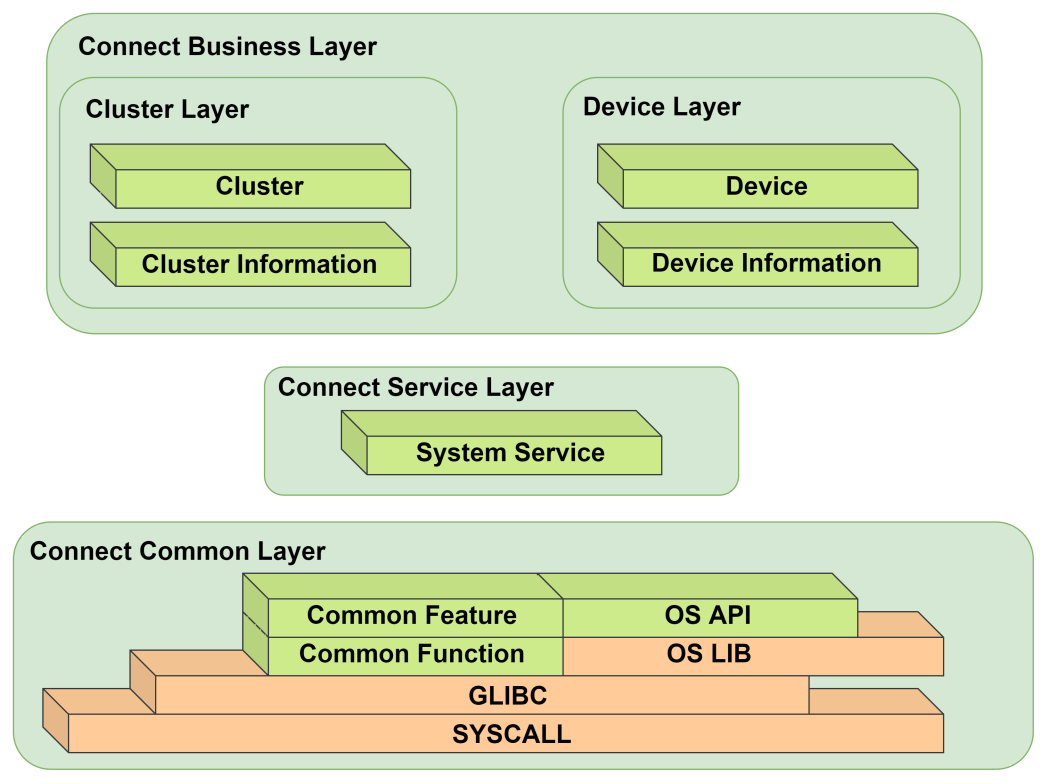
## 功能模型

### 功能层次模型

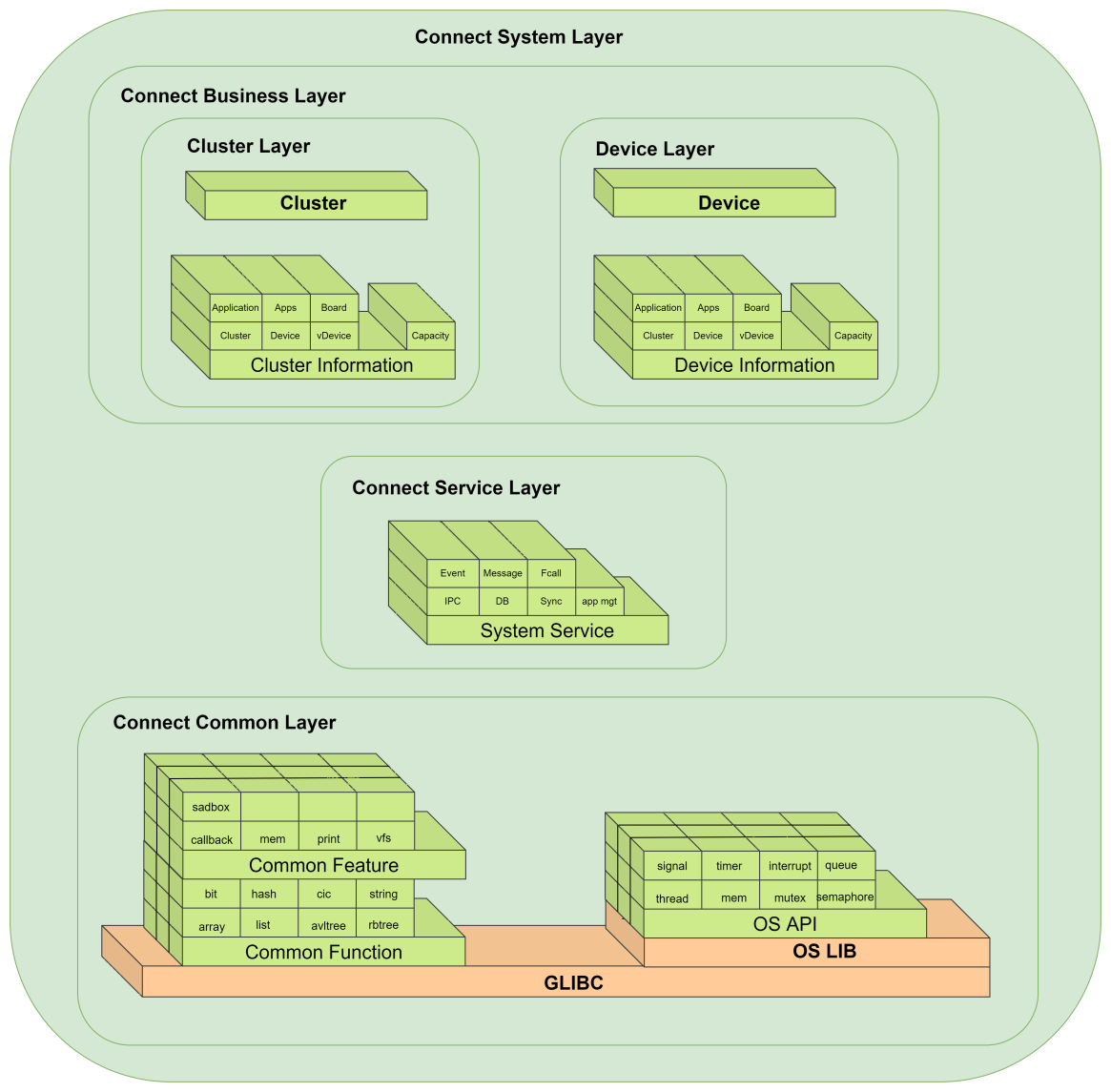
业务层次模型主要反应业务的层级关系以及业务在模块之间的流向，功能层次模型主要反映业务模块借助内部的功能模块实现业务的依赖关系，业务层次是对外可见的层次结构，而功能层次则是内部的，在业务层面下面的层次结构，



按照功能层次划分系统可以分为业务层，服务层和通用层，业务层主要是设备对外提供的业务，包括网络业务，设备业务和系统业务（纵向划分），服务层主要为各种业务提供系统服务，服务一般有进程间的服务也有进程内的服务，比如IPC，数据库服务等，通用层主要是通用的代码组件，业务功能和服务功能都可以很方便的借助通用代码快速实现功能业务。



详细的层次结构分析如下，下图只是例子不完全包括所有业务，



### 系统服务

系统服务和系统业务都是在系统平面对其他业务提供服务，它们之间的区别在于系统服务的内容不对外呈现。

系统服务是系统的基础，系统服务的稳定性，易用性和灵活性很大程度影响系统的稳定性和扩展性，下面章节介绍几个主要的系统服务。

#### IPC业务

IPC是进程之间通信的基本形式，相比其他进程间通信（共享内存，数据库）这种方式最符合业务逻辑思维，可以使业务逻辑保持简单。

传统的IPC模型是1对1收发模型，但是对于系统中进程数量越来越多如果每两个进程之间都建立IPC连接那么N个进程将总共创建N\*N条IPC连接数量太大。考虑到计算机系统中的总线通信也有类似情况，在这种情况下总线一般支持Switch用于各个终端之间的通信转发，所以面对IPC连接数量太多的问题，IPC将支持IPC switch服务用于IPC数据在进程之间的转发。

#### 数据库

采用数据库可以使配置数据和代码逻辑分离支持进程重启业务，从而使系统整体上更加稳定，同时由于进程可以重启所以可以支持系统热升级。另外各个层级的业务通过数据库获取配置数据和状态数据更加集中和容易，避免了过多的IPC接口调用。

数据库作为Connect系统中的服务业务对Business层中的四个子层（Cluster Management，Cluster Control，Device Control，Board Control）都提供数据库业务，另外也对服务层提供数据库业务，

Cluster Management Database

Cluster Control Database

Device Control Database

Board Control Database

Service Database

#### 调试服务

调试一般由调试命令，调试打印等多种调试手段组成。

对于调试命令如果把其放到命令行系统的调试模式对模块来说层间的接口会比较多，因为会针对调试命令增加许多层间的IPC接口带来额外的代码复杂度。最理想的做法应该是通过特殊的调试命令行系统借助进程管理，在调试命令行和进程之间建立一条单独的通道，调试命令行通过这条通道直接调用进程内置的命令进行调试，而DEBUG模式下的调试命令更像Tech support的功能。

另外一个比较重要的调试功能是调试打印，通过打开某个模块的调试打印分析业务的运行情况，调试打印同样需要将进程的输出重定向到控制台。

GDB调试属于研发调试手段不在本设计文档中描述。

### 通用功能

通用功能主要是一些通用数据结构，通用库和系统封装接口，这些功能都是以库的形式存在。

不同的通用功能的使用范围有点区别，一般来说如果是通用特性，比如内存管理等都是基于进程有效的，而如果仅仅是通用功能，比如数据结构则是基于业务初始化单独的控制信息有效。

## 业务关系

业务之间的关系有业务依赖，系统业务和系统服务。

### 业务依赖

业务依赖都和需求有关。

#### 业务需求

业务依赖从业务需求开始，业务组件相互配合实现业务需求从而产生业务依赖，这种依赖一般都是跨层的依赖，比如管理层服务依赖控制层业务提供的接口。

#### 业务作用范围

业务的最终目的是为了实现需求，而需求总是有其作用的对象和范围，业务需求存在天然的作用域依赖。

##### 网络业务作用域

基于网络层次模型常见的网络作用域有，链路层端口（包括trunk），VLAN+端口，VLAN，VLAN-IF，子接口等，这些作用域具有能力，使能，转发状态以及Link状态等属性，由于业务基于这些作用域实现，所以当作用域的属性发生变化总是会影响到相关业务。

对于能力属性，如果能力新增相关业务对应增加作用域以及初始化，业务根据具体情况决定是否运行；如果能力移除，业务移除对应配置并停止相关作用域业务，由于配置删除对用户来说不是特别友好，能力移除配置直接删除还是保留配置让业务停止运行有待考虑。

对于使能属性，如果使能打开并且业务使能也是打开状态那么业务运行；如果使能关闭，业务停止运行。

对于转发状态属性，如果允许转发并且业务使能也是打开状态那么业务运行；如果转发状态处于禁止或者其他属性，那么业务停止或者暂停。

对于Link状态属性，如果处于Link UP状态并且业务使能也是打开状态那么业务运行；如果处于Link Down状态，那么业务停止或者暂停。

作用域属性存在能力属性变化影响配置属性，配置属性变化影响状态属性的传递现象。

另外由于作用域由于网络层次结构的关系，有下层作用域变化引起上层作用域变化的转递现象。

##### 设备业务作用域

设备业务的作用域有簇群，设备和板卡（可能由子卡），业务的作用域是簇群时，呈现的数据往往是虚拟的和经过逻辑仲裁的，而作用域时设备和板卡时一般和实际的设备硬件关联。

簇群，设备和板卡是从大到小包裹的关系，所以他们的属性变化产生的影响存在从大到小的传递关系。

##### 系统业务作用域

系统业务的作用域有簇群，设备，板卡，进程和模块，对于簇群，设备和板卡往往是抽象的概念最终只能是板卡，进程和模块的集合。

#### 实现需求

业务要实现需求可能有多种不同的方式，某些方式可能对其他业务存在资源要求，从而产生资源依赖；此外业务自身在特定的系统环境中运行，由于硬件资源以及其他因素的影响存在天然的硬件能力和环境规格依赖。

对于网络业务一般对芯片的V/I/E CAP，L2 Table，L3 Table，XSLT Table等有资源占用需求。

对于设备业务一般对控制器有资源使用需求。

系统业务没有明显的实现需求。

### 系统业务 vs 系统服务

系统业务和系统服务的界限很小，本质上它们都是系统层的功能，我们把它们分开的唯一原因是系统业务会呈现给用户，而系统服务作为内部使用，系统业务和系统服务的作用域和层间依赖都比较相似。

## 初始化

### 进程初始化

如果没有特别的原因进程必须支持重启，进程支持重启的前提条件是：

1. 配置分离

这点通过数据库模型可以支持。

1. 数据恢复

根据数据类型的不同，业务层级的不同数据的恢复手段也不同，

能力配置通过重新读取能力静态库或者数据库中的能力数据恢复；

配置数据应该通过重新读取数据库中的配置数据恢复；

1. IPC分离

即进程退出或者未初始化完成前如果有其他进程通过IPC访问该进程不会导致双方出现严重后果，这点需要通过IPC switch支持。

1. 业务逻辑解耦

进程之间不应该出现强依赖关系。

进程初始化按照如下几步进行，

1. 进程自身的资源和基本信息初始化；
2. 服务模块对接初始化，主要是一些提供被动服务的业务，比如日志，告警，打印等；
3. 向IPC注册开放访问其他进程的能力；
4. 进程内的服务初始化；
5. 数据库访问初始化；
6. 模块读取数据库配置依次初始化；
7. 服务模块对接初始化，主要是一些主动服务业务，比如事件，消息等；
8. 向IPC注册开放被访问能力；
9. 打开报文收发能力；
10. 打开进程内业务功能。

### 设备初始化

嵌入式系统的初始化流程不变，uboot启动，kernel启动，待kernel启动完成我司的业务从init.d初始化脚本开始启动，

1. 内核模块注入；
2. 系统服务启动（系统服务的配置属于内部配置不对外呈现）；
3. 配置数据导入数据库；
4. Cluster控制业务启动（包括网络，设备和系统）；
5. Device控制业务启动（包括网络，设备和系统）；
6. Board控制业务启动（包括网络，设备和系统）；
7. Cluster管理业务启动；
8. 激活

### 簇群初始化

设备初始化完成以后如果配置了簇群业务，则有Cluster控制层的Cluster协商业务负责簇群发现，协商和加入。

当设备加入簇群，如果设备作为主设备，Cluster控制层的业务继续保持运行，如果作为备设备Cluster控制层业务保持静默，如果主设备异常则有从设备接替运行，如果设备作为成员则关闭Cluster控制层业务。

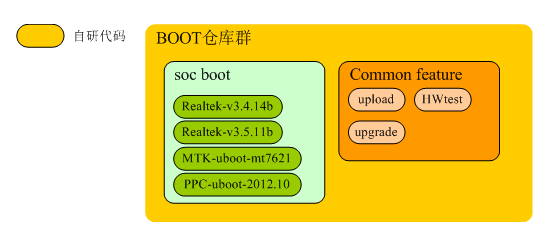
## 代码组织结构

### 代码组织

根据业务模型和功能模型分析，可以将代码进行划分。

#### BOOT

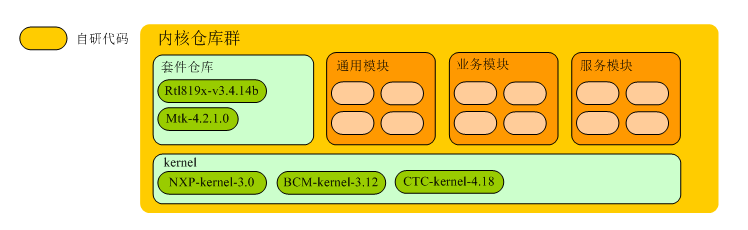
对于网络设备产品一般会有单独的boot代码，虽然不同芯片厂家对自家的芯片使用的boot不尽相同，但是集成厂家一般会增加一部分自身的通用代码，因此boot的代码可以组织为，



从上图可以看出，BOOT仓库群由不同厂家的boot代码和本公司的功能代码组成，不同厂家对不同芯片提供的每一套不同版本的boot代码都作为一个独立的部分进行维护。

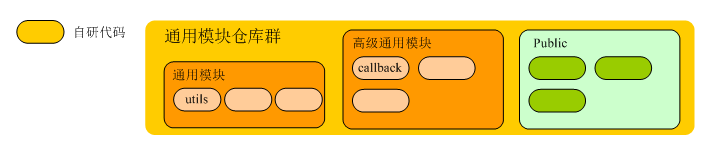
#### 内核

目前的设备主要使用Linux系统，一般来说CPU芯片供应商会提供参考的Linux内核版本，集成厂家基于芯片厂家提供的版本进行开发和维护，



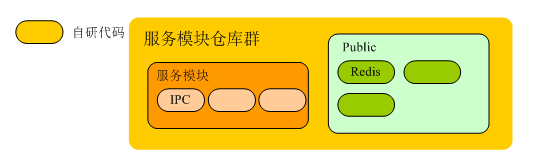
#### 通用模块

除了上一章节的代码属于内核部分，从本章开始都是讲述用户态下的代码。通用代码顾名思义即可以跨越多种不同产品线不同芯片方案使用的代码，通用代码的抽象程度较高有时需要产品或芯片相关代码配合才能使用，



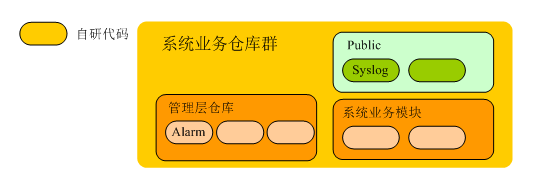
#### 服务模块

服务模块可以为某个模块服务，可以为某个进程服务，也可以为整个系统服务，



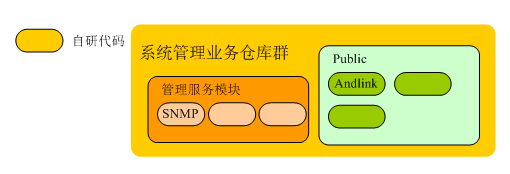
#### 系统业务

系统业务一般情况是基于整个系统，为多个进程中的多个模块同时提供服务，同时自身也提供业务配置能力，



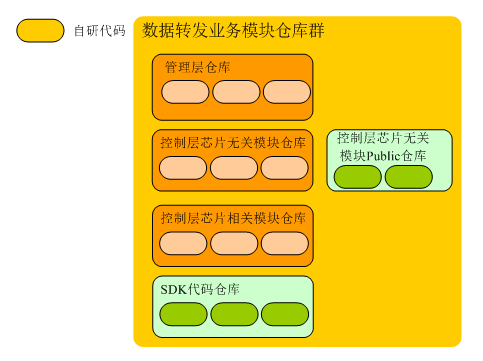
#### 系统管理业务

系统管理业务主要是业务功能的管理层的服务提供模块，



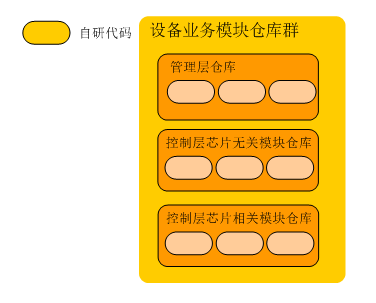
#### 网络业务

作为通信产品，网络业务是其核心业务，



#### 设备业务

设备业务是每个设备的基本功能，



### 组件包

考虑到管理的粗细粒度以及复杂度，软件以组件包的形式进行开发和维护，

#### 组件构成

组件由一组进程以及相关的配置文件构成，进程的数量和组件自身的设计有关，而配置文件主要用于组件中进程的启动引导，默认配置等相关业务。

组件内的代码组织和配置文件组织受组件自身设计限制，但是每个组件有一些基本的约束需要遵守，以便整个编译，打包和安装能够正常进行。



一个典型的组件包如上图所示。

code子目录用于存放组件包代码，图中只有code目录但实际在设计过程中可以有多个code目录。

Include子目录用于存放组件包对外开放的数据定义和接口，一般情况下只有组件包是动态库时才需要对外开放头文件。

image子目录中的所有内容将被拷贝到设备文件系统的根目录下，图中只有etc目录，一般情况下etc目录中会放置组件包的启动脚本和配置文件，但是根据组件包自身的设计在image目录下还可以放其他子目录，比如usr，opt，lib等，需要注意的是image下的目录最好都遵循linux文件目录命名规范，不要擅自取名字。

DEBIAN子目录用于组件包dpkg安装，典型的DEBIAN目录中有以下文件，



在组件包的根目录下一定包含组件包编译的MAKEFILE文件。

#### 编译

组件包的编译由外部编译框架引导编译，“make -C 组件包/”。

#### 版本控制

考虑到组件包功能的持续演进以及当前设备的稳定性，需要对组件包进行版本控制。组件包的版本控制目录如下，



一个组件包有上图所示中的目录。

Trunk目录为组件包的主干版本，主干版本不允许研发代码直接提交。

Develop目录为项目开发目录，当一个项目开始时从trunk目录拉出develop目录进行开发，开发过程中的代码都提交到develop目录中。

Branch目录为特别分支目录，当功能出现较大分歧时需要通过trunk开出branch分支进行单独演进。

Maintain目录为项目维护目录，当研发部门在develop目录中完成单元测试和集成测试并通过了测试部门的准入测试，这时develop目录中的代码将合入trunk目录，然后从trunk目录中拉出maintain目录进行系统测试，后续对BUG的修订都提交到maintain目录中。Maintain目录中的BUG修订周期性的同步回trunk主干。另外产品发布后长期的BUG修订和版本发布都在maintain目录中进行。

Tag目录主要用于标记项目节点，当系统测试通过软件版本允许发布时基于当前的maintain目录会打出相应的tag目录。

Custom目录主要用于客户定制的临时版本。

典型的组件包目录如下，



#### 升级

组件包采用标准的Linux DEBIAN格式，可以通过dpkg升级。

考虑到有些基础服务的依赖特性，组件包升级应该做依赖检查。

## 编译和打包

### 传统编译打包

一个全新的嵌入式项目首先应该在电脑主机平台上通过电脑主机平台的编译器把目标平台的编译器编译出来，有了目标环境的编译器才能编译目标板卡上的所有程序。

一个嵌入式系统在业务系统运行之前一般有一段boot程序提前运行，boot主要用于硬件初始化，硬件检测以及当业务系统异常时可以在boot阶段刷新系统修复，boot和业务系统是两个完全独立的系统，当然两个系统之间也不是完全隔离，比如uboot和Linux之间就会通过dtb传递配置信息。

有了编译器，第一步应该先把boot编译出来进行板卡调试和硬件验证，随后通过配置boot引导业务系统。

业务系统的编译流程一般会按照内核，通用库和业务模块的顺序进行编译，内核内部的功能以及业务模块之间的顺序也会按照各自的设计进行相应的安排。编译内核时，首先通过内核配置文件控制要编译的功能，比如Linux有自身的配置文件进行控制；而通用库的编译大部分由芯片供应商在前期进行配置和调试，集成商可以根据芯片供应商的方案进行调整；最后业务模块通常由集成商自行定义依赖关系并配置编译顺序。

当所有代码都编译完成，根据芯片供应商或者集成商自己的方案对编译完成的程序进行打包和加密。

完整的嵌入式编译流程应该是首先通过开发环境编译目标环境编译器以及配套组件，再通过目标环境编译器编译boot，Linux系统，标准库，通用组件和业务组件，然后把标准库，通用组件和业务组件制作成文件系统，最后把boot，内核映像和文件系统映像一起打包制作成升级文件。

Connect系统作为业务系统中的业务模块进行编译，Connect内部采用代码集中编译的方式进行。

### 组件化编译打包

后续Connect系统业务部分的代码将由集中编译初步拆分为组件化编译。

### 编译打包工程版本控制

编译工程的版本控制分为几个部分，

对于通用功能仓库根据新增的业务功能进行演进，每次某个或者某些功能需要新增或者更新都需要新的develop分支，当功能开发并验证完成后develop分支转换成maintain分支进行长期维护同时销毁develop分支，另外如果有方案仓库从maintain分支或者trunk分支开出，那么需要基于此点进行tag分支创建。

对于方案仓库来说由于一个芯片方案可能有多个项目，这些项目立项后可以从trunk分支或者相似maintain分支开出自己项目的develop分支，当项目业务组件进入maintain对应的develop分支也开出maintain分支进行维护。总之方案仓库的分支管理应该和组件的分支管理类似。

## 主要组件

*后续对代码详细分析以后在这里列出*

## 软件内存估计

略

## 性能

从目前分析来看软件组件应该没有特别的性能要求。

# 问题，风险和依赖

## 平台需求

需要搭建新的平台。

## 相关项目

暂无。

## 第三方关联

暂无。

## 唯一来源的组件

## 技术需求

暂无。

## 技术风险

暂无。

# 附录A-标题

无。

# 需求可跟踪考虑

参考《交换机 需求规格说明书》和《交换机 需求规格》，我司所有的需求ID都是唯一，方便和项目后续的设计文档对应，以及方便系统测试对应验收。

# 参考资料

《交换机 需求规格》

《交换机 需求规格说明书》

# 术语表

略。

# 附件

## 评审活动记录