硕士学位论文

（工程硕士）

基于 OpenWrt 平台的路由器软件系统的设计优化与实现

DESIGN OPTIMIZATION AND IMPLEMENTATION OF ROUTER SOFTWARE SYSTEM BASED ON OPENWRT PLATFORM

张一弓

哈尔滨工业大学

2013 年 6 月

国内图书分类号：TP315　　学校代码：10213
国际图书分类号：621.1　　密级：公开

工程硕士学位论文
（工程硕士）

基于 OpenWrt 平台的路由器软件系统的设计优化与实现

硕 士 研 究 生：张一弓
导　　师：宋颖慧副教授
副　导　师：郑淇文高级工程师
申 请 学 位：工程硕士
学 科 、 专 业：软件工程
所 在 单 位：软件学院
答 辩 日 期：2013 年 6 月
授予学位单位：哈尔滨工业大学

Classified Index：TP315
U.D.C: 621.1

Dissertation for the Master's Degree in Engineering

DESIGN OPTIMIZATION AND IMPLEMENTATION OF ROUTER SOFTWARE SYSTEM BASED ON OPENWRT PLATFORM

Candidate: Zhang Yigong
Supervisor: Associate Prof. Song Yinghui
Associate Supervisor: Senior Engineer Zheng Qiwen
Academic Degree Applied for: Master of Engineering
Speciality: Software Engineering
Affiliation： School of Software
Date of Defence: June, 2013
Degree-Conferring-Institution: Harbin Institute of Technology

摘　要

众所周知，路由器一直是计算机网络的核心设备，随着互联网取得的巨大成功，路由器也在不断扩大自己的自身的使用范围。目前路由器设备自身发展非常迅速，相信随着通信行业整体的发展，路由器技术也会更加完善并且稳定，将会给用户带来更加良好的网络环境。因此，研发出一款小型家用多功能的无线路由器产品，不但具有很高的技术研究意义，而且具有很高的市场价值。路由器功能与性能的强弱主要取决于两个方面，一方面是由路由器自身硬件参数所决定；另一方面则取决于路由器软件操作系统的稳定性、安全性、兼容性、易操作性、工作效率等，这就需要设计人员设计好整个路由器软件系统架构。

本论文的主要研究成果为：通过分析和研究原有开源的 OpenWrt 路由操作系统的架构，以及 OpenWrt 平台存在的缺陷，结合软件工程的原理和思想，设计优化和实现原有 OpenWrt 软件系统，并把它配置到本项目蝴蝶路由器中。主要包括新设计的消息总线模块、服务管理模块、配置管理模块。在软件的设计过程中采用模块化的思想，达到了低耦合的效果；在消息总线模块的设计实现中，通过查阅相关技术文档与文献，主要使用命名管道的方法实现了进程间通信，取代了原有 OpenWrt 平台使用桌面级消息总线 D-Bus 结构过于臃肿与系统资源开销较大的缺陷；同时，在服务管理模块中，使用对 Linux 下/proc 目录定时遍历和父进程主动上报的方法实现了对系统中子进程的完全监管；另外，设计并实现了服务管理模块与配置管理模块，弥补了原有 OpenWrt 没有服务管理的功能和配置文件轻易读写、操作的缺点。

最后，通过系统各模块测试和产品的整机测试，该路由器软件系统功能点齐备，路由器 WEB 配置界面跳转流畅，通过界面的配置修改请求，可正确完成路由器系统的各个配置修改。总体性能满足客户要求。

关键词：无线路由器;消息总线；配置管理；OpenWrt 优化

Abstract

As we all know, the router has been the core of computer network equipment, along with the great success of the Internet, the router is also expanding the use of their own. Currently router device itself is developing very quickly, I believe that with the development of the communications industry as a whole, router technology will be more perfect and stable, it will give users a more favorable network environment. Therefore, the development of a multi-functional small home wireless router products, not only has high technical significance, but also has a high market value. Router functionality and performance depends primarily on the strength of two aspects, one is the hardware parameters from the router itself is determined; the other hand, depending on the router software operating system stability, security, compatibility, ease of operation, operating efficiency, which requires designers to design the entire router software system architecture.

The main results of this dissertation are: the analysis and research of the original open-source operating system OpenWrt routing architecture, as well as defects OpenWrt platform that combines software engineering principles and ideas, design optimization and implementation of the original OpenWrt software systems, and to it configures the router to the project butterflies. Including newly designed message bus module, service management module, provisioning management module. In the software design process using modular thinking, reaching low coupling effect; in message bus module design implementation, through access to relevant technical documentation and literature, the main use named pipes method to achieve inter-process communication, replacing the original OpenWrt platform desktop-class D-Bus message bus structure is too bloated and system resource overhead larger defects; while in the service management module, use the Linux under the /proc directory traversal and regularly reported to the parent process proactive approach to achieve a neutron processes on the system is fully regulated; addition, the design and implementation of service management module and provisioning management module, to make up for the original OpenWrt no service management functions and configuration files to easily read and write operations shortcomings.

Finally, through the system of the whole module testing and product testing, the router software system function points are available, the router WEB configuration interface jump smoothly through the interface configuration modification request can be completed correctly each router system configuration

changes. Overall performance to meet customer requirements.

Keywords：Wireless router；Message bus；Configuration management； OpenWrt optimization

目　录

第 1 章 绪 论

1.1 课题的背景及研究目的

众所周知，路由器一直是计算机网络的核心设备，随着互联网取得的巨大成功，路由器的使用范围在不断扩大。在企业、商场、学校、机关、娱乐场所到处可见路由器的身影。随着无线技术的发展，以及个人终端设备（手机、上网本、笔记本、个人 PC 等）上网功能的提升，咖啡馆、快餐厅、住宅区等人们活动的场所，到处可以找到无线网络信号。因此，人们对于小型高端无线路由器的需求也在不断增大，路由器也在不断扩大自己的自身的使用范围。目前路由器设备自身发展非常迅速，相信随着通信行业整体的发展，路由器技术也会更加完善并且稳定，将会给用户带来更加良好的网络环境。因此，研发出一款小型家用多功能的无线路由器产品，不但具有很高的技术研究意义，而且具有很高的市场价值。路由器功能与性能的强弱主要取决于两个方面，一方面是由路由器自身硬件参数所决定；另一方面则取决于路由器软件操作系统的稳定性、安全性、兼容性、易操作性、工作效率等[1]，这就需要设计人员设计好整个路由器软件系统架构。

OpenWrt 是一款开源的路由器操作系统，使用 LINUX 内核，其功能十分强大。OpenWrt 被描述为一个嵌入式设备的 Linux 发行版，而不是试图建立一个单一的静态固件，OpenWrt 的包管理机制提供了一个完全可写的文件系统，并允许开发人员自定义设备。在当今路由器市场上，国内外路由器厂商大多均选择 OpenWrt 作为自己路由器产品的操作系统，蝴蝶路由器也将采用 OpenWrt 作为自己的软件系统。但是，OpenWrt 在系统架构上并不满足蝴蝶路由器产品的要求，因为，OpenWrt 自身的系统架构过于松散，各模块之间的联系不够紧密，消息机制过于臃肿，配置管理等问题上还存在着缺陷。这就需要设计人员在研发自己路由器产品的时候，要针对自身产品的需求来对 OpenWrt 操作系统进行修改、裁剪、优化，重新设计 OpenWrt 系统的架构，使其符合自己路由器产品的要求。本课题研究的目的和意义就在于通过修改并且重新定制 OpenWrt 系统，实现对 OpenWrt 的优化和改造，以及实现对蝴蝶路由器软件系统架构的良好设计，使其软件系统不但有更高的工作效率，而且更加稳定，来满足蝴蝶路由器产品的需要。

1.2 与课题相关的国内外研究综述

1.2.1 国内外路由器操作系统举例

路由器操作系统是管理路由器硬件设备与软件资源的程序[2]。路由器操作系统是控制其他程序运行，管理系统资源并为用户提供操作路由器界面的系统软件的集合[3-4]。路由器操作系统负责管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本事务[5]。目前国内外常见的路由器操作系统有 Vyatta、ClearOS、RouterOS、ZeroShell、OpenWRT 等。

Vyatta 是一种完善的 Linux 发行版，它能够将标准的 X86 硬件平台转换成路由器。Vyatta 还会提供对工业标准路由协议、常用网络扩展接口以及管理协议的支持。所有的这些功能特性均可以通过命令行输入和 WEB 配置用户界面来进行配置操作。对个人操作系统而言，Vyatta 更多的是用在家庭的路由器和防火墙上，特别是商业驱动版本 Vyatta 能够很好地支持的通信需求。Vyatta 路由软件可以通过开放的自由社区获得，Vyatta 也提供付费的项目服务，服务将包含软件产品的维护、升级和技术支持。

ClearOS 基于分布式环境而设计，是一款出色的路由系统，具备现有路由系统的大部分功能，如 DHCP，端口转发，防火墙等。同时因为它基于 Red Hat（一款 Linux 发行版），能提供良好的功能扩展支持。主要面向中小企业和分布式环境而设计的网关和网络服务器。

RouterOS 路由操作系统软件将标准的 PC 机转化为专业路由器，RouterOS 在软件开发和应用领域的不断更新和发展中，软件已经经历了很多的更新和改进，使其功能在不断增长和改善。特别是在无线、用户认证、带宽控制、策略路由管理，和防火墙过滤等功能方面有非常突出的特点。RouterOS 路由器操作系统具有很高的性价比，这一点得到了许多用户的青睐。RouterOS 添加标准的网络接口卡能增强路由器的功能。一台个人计算机就可以实现路由功能，提高硬件性能同样也能提高网络的访问速度和吞吐量。完全是一套低成本，高性能的路由器系统。

ZeroShell 也是一款 Linux 发行版，将它装入到个人电脑、服务器和嵌入式设备上。除了提供基本的局域网服务外，它还提供实现企业级无线安全的远程拨号服务器、带反病毒功能的 HTTP 代理服务器、OpenVPN 服务器及客户端，以及强制网络门户等服务。其他的一般功能包括可对多个互联网连接实行负载均衡和故障切换，以及无线接入点模式，并支持多个

SSID（Service Set Identifier）和虚拟局域网。

1.2.2 OpenWrt 平台的国内外研究状况

WRT54G 是路由器中的经典产品，由思科公司在本世纪初研制发行，它是基于 GNU（General Public License）的 Linux 操作系统。同期，思科公开了 WRT54G 的源代码。至此，这款路由器系统便有了长足的发展。单兵作战及团队合作促使许多业界人士都贡献自己开发成果，极大促进了这款路由器的更新速度，从此该操作系统诞生，命名为 OpenWrt。它有包管理机制、直接编译到重新编译、文件重写等优点。目前，OpenWrt 已成为各大厂商的主流开发平台，基于 Linux 内核、免费是其它软件产品无法替代的优势。目前为止，OpenWrt 已经编译 100 多种软件产品，并且数目还在不断增加。而 OpenWrt SDK 开启了更简洁开发的序幕。各厂商均在按着自己不同的需求来对 OpenWrt 系统进行改造，从而设计自己的路由器产品，如果 Ubuntu 是桌面版的 Linux 系统，安卓是手机版的 Linux 系统，那么 OpenWrt 可以称之为是路由器版的 Linux 系统[6]。

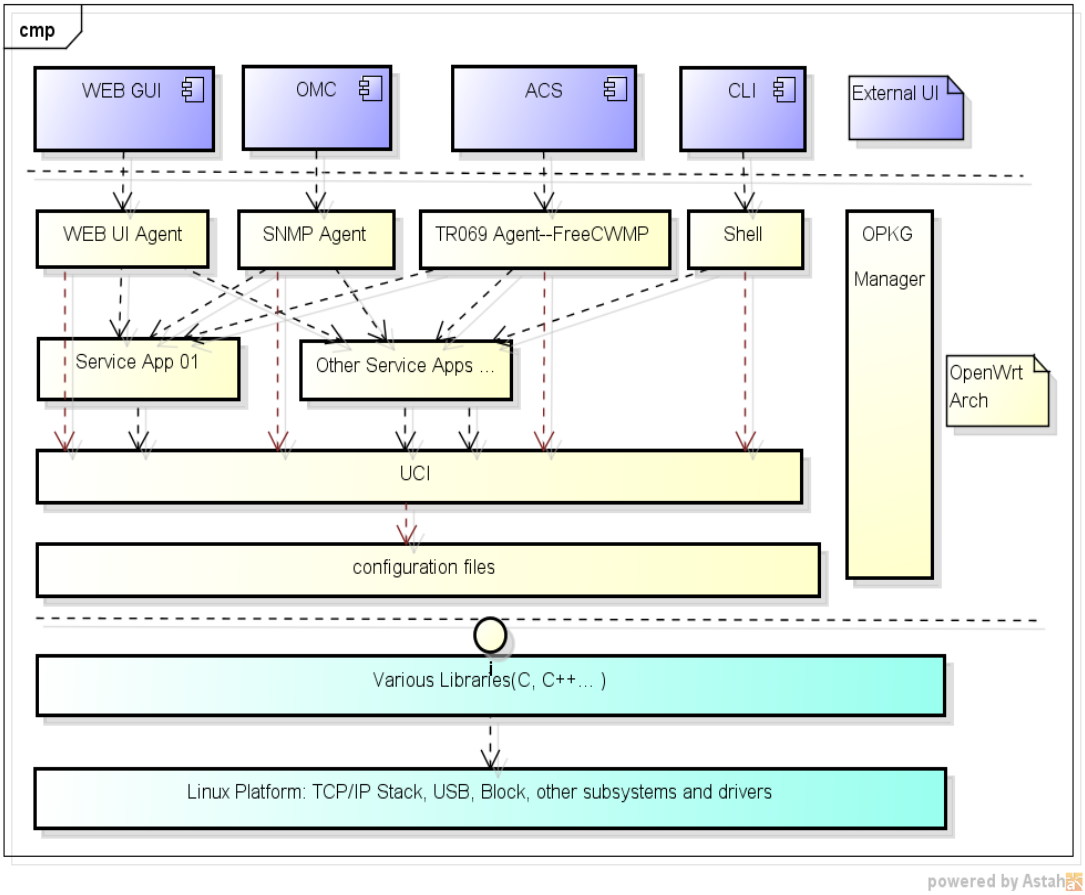

图 1-1　OpenWrt 平台架构

OpenWrt 作为一个开源项目，随着近些年 OpenWrt 的发展，OpenWrt 提供

了许多与商用家庭网关固件相同的功能，如 DHCP 服务、WEB 无线加密。它还提供了商用固件没有或者效果不好的众多功能：端口映射，使外网能访问内网特定服务；UPnP 即插即用，可动态配置端口映射；静态 DHCP 可扩展防火墙和路由器配置 QOS（Quality of Service）应用，如网络电话，在线游戏和流媒体 ；具有无线中继器，无线接入点，无线网桥的功能，甚至可结合这三种功能一起使用；网状网络 DDNS，支持动态 DNS 服务，命令行可以通过 SSH（Secure Shell）或 Telnet 远程登录；通过设备上的 USB 端口，可支持打印机共享、Windows 文件共享、FTP 服务器、USB 音频以及几乎任何可通过 USB 连接的其他设备实时网络监控 ；LUCI（Lua Unified Configuration Interface）WEB 配置界面。

OpenWrt 的基本架构如图 1-1 所示，OpenWrt 使用统一软件包管理，系统的每个功能都是由每个具体的软件包所实现，可以通过增加和删除软件包来实现对系统功能的增加与删减，最终达到系统所要功能需求[7]。位于系统的第一层为外部应用程序。OpenWrt 平台所支持的外部应用程序包括 Web GUI（Graphical User Interface）、OMC（Operation Maintenance Center 简单协议管理端）、ACS（Access Control Sever　TR069 服务协议）、CLI（命令行控制终端）。OpenWrt 所用的 WEB 程序叫做 LUCI，LUCI 是一个 WEB 框架，也是 OpenWrt 上默认的 WEB 系统。LUCI 是通过 Lua 脚本语言生成动态页面，以及 UCI 访问系统配置文件中的数据而完成整个工作的，也可以说 LUCI 是由 Lua 语言和 UCI 共同实现的[8]。LUCI 是用一种叫做 Lua 的轻型脚本语言所编写，Lua 使用面向对象的库和模板，确保了高效的执行，轻量的安装体积，更快的执行速度以及更好的可维护性。LUCI 即是这两个项目的合体[9]，OpenWrt 上的默认 WEB 系统。各路由器厂商通过 Web GUI 界面的自定义，实现各自路由器人性化的用户访问界面，从而实现了通过 WEB 页面来对路由器进行配置。SNMP（Simple Network Management Protocol,）是简单网络管理协议，由一组网络管理的标准组成，包含一个应用层协议、数据库模型、以及一组资料物件，OMC 是其服务器端应用程序。TR-069 被称为 CPE 广域网管理协议，在 OpenWrt 系统得到了广泛应用。它的目的是为了实现路由器的零配置，路由器可根据此协议，由电信服务商直接为本地路由器实现配置。命令行界面 CLI（Command Line Interface）是所有路由器产品提供的外部界面，如思科、中兴、华为等。它是路由器产品的标准。但是目前市场上有些产品为了降低成本，不再使用 CLI。使用 CLI 只需要串行口，而且功能扩充非常方便，若有多台路由器需要一同进行配置，路由器的配置可以通过 COPY 与 PASTE 功能很快完成。不需要通过

WEB 浏览器每一台都进行配置。Web GUI、OMC、ACS、CLI，以上四个外部 UI 已经成为各路由器厂商在定制 OpenWrt 平台时默认要求支持的外部访问程序。

系统的第二层是各外部接口对应的解析模块，由 WEB UI 解析模块（WEB UI Agent）、简单管理协议解析模块（SNMP Agent）、TR-069 服务端解析模块（ACS Agent）、命令行解析模块（CLI Agent）组成。UCI 是 OpenWrt 所提供的通用配置接口，各解析程序通过调用 UCI 提供的配置文件读写接口来修改配置文件。位于操作系统的最底层是、驱动、协议栈、Linux 内核等。各个模块间通过消息机制进行通信，将各个模块之间联系起来[10]。OpenWrt 使用 D-Bus 消息总线实现了进程间的通信。

D-Bus 使用 GPL（General Public License）许可证发行，是一种高级的消息总线。D-Bus 的主要用途是在 Linux 桌面环境下为提供进程间通信功能。同时，D-Bus 消息总线能将 Linux 桌面环境和 Linux 内核事件作为消息传递到进程[11]。在消息总线上注册过的进程可通过总线发送或接受消息，各个进程也可在消息总线上注册后，监听并等待内核事件响应。目前，D-Bus 消息总线已经被绝大多数 Linux 发行版采纳并使用，各种复杂的进程间通信作业均可使用 D-Bus 消息总线来实现。D-BUS 的功能已经覆盖了所有进程间通信的需求，虽然 D-Bus 是应用于 Linux 桌面环境下的进程间通信，但是 OpenWrt 依然选择了 D-Bus 作为路由器系统的消息处理方式[12-13]。

OpenWrt 平台没有实现统一的配置管理。OpenWrt 平台使用的配置文件管理方法是用 UCI 作为统一的配置访问工具。在 OpenWrt 原有的设计和实现中，各个外部 UI agent 模块都是通过 UCI 的接口直接操作配置文件。WebGUI agent 使用 Lua UCI 的接口，TR069 agent，SNMP agent 以及 CLI agent 使用 CLI UCI 的接口。同时各个服务进程也是通过 Lib UCI 的接口读取各自的配置文件。OpenWrt 使用 UCI 加 Shell 的方式统一了配置的接口，但是这些配置操作管理都是在不同的解析模块下的上下文所调用的[14]。在多个文件需要修改的时候，虽然 UCI 可以使用文件锁对配置文件进行操作，但是多个解析模块之间对于配置文件变更的同步性比较弱，或者说原有的架构就是不能及时的同步彼此的修改的。华为于 2009 年发布的一款基于 OpenWrt 平台的路由器解决了这个问题。华为所使用的方式是通过增加配置管理模块。配置管理模块将所有解析模块对 UCI 的调用动作集中管理。各解析模块直接把要修改的信息重定向到配置管理模块，由配置管理模块统一完成后续配置修改工作。华为在后续其他基于 OpenWrt 平台的路由器产品设计中，均一直延续了该方法。

系统中的服务是由一个应用程序进程或多个应用程序进程所组成。OpenWrt 平台没有实现对服务的监管，各服务均孤立的存在。思科在路由器系统服务管理功能上的设计一直处于行业内领先地位。思科所采用的设计方法是增加相应的应用程序管理模块，该模块的作用是统一系统内各应用程序的启动与关闭操作，以及对系统服务中注册、卸载、启动、关闭、重启操作的统一管理。

1.2.3 OpenWrt 平台消息总线研究状况

消息总线模块作为路由器软件系统的基础软件模块，负责各模块间的消息转发和消息广播，向整个系统提供 C API 接口和命令行接口。OpenWrt 平台采用的 D-Bus 消息总线在业内一直得不到很高的评价，原因在于，D-Bus 是一款面向桌面环境设计的消息总线。D-Bus 的 API 接口非常丰富，对其数据结构的定义非常庞大。D-Bus 把定义好的接口与相关结构用支持库的形式存在，在系统运行时，将支持库的全部内容均加载到内存当中。而嵌入式设备无法容忍像桌面环境下如此庞大系统开销[15]。因为，在嵌入式设备中内存资源十分有限。在路由器系统中仅需要一些简单的接口，包括：消息的注册、消息的注销以及消息的发送与接收。而无须桌面环境下其他扩展功能。其他在桌面环境下的扩展功能对路由器环境下没有任何作用。而且，路由器设备也不需要桌面环境下庞大的数据结构定义。这些，均增加了系统额外开销与模块设计难度。因此，大部分路由器厂商均各自开辟新道路，来实现自己路由器产品的进程间通信方式。多数公司在这个方面的做法是直接修改 D-Bus 开源代码，对数据结构重新定义，然后对函数库进行裁剪，仅保留例如消息注册、发送、接收等最基本的接口。

2012 年 6 月，OpenWrt 在官网上发布了非正式版的 U-Bus 消息总线。U-Bus 是专门为嵌入式平台设计的，它以一种“微总线”的方式来提供进程间通信功能，系统开销十分小。各路由器厂商对这种新总线产生了浓厚的兴趣，而且，国内市场上还未出现过使用 U-Bus 的路由器产品的报道。预计在未来的路由器设计中，使用 U-Bus 消息总线来实现进程间通信是一个共认的方向。

U-Bus 的目的是为了完成消息的转发，实现各个模块之间的通信。也可以说，是实现了两个进程之间的通信。总线是 U-Bus 的进程间通信机制，如图 1-2 所示：

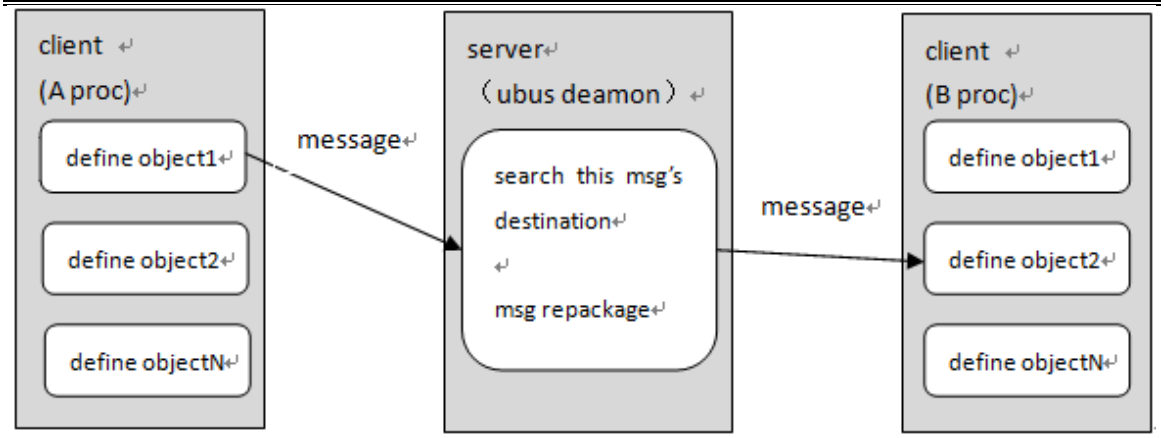

图 1-2　U-Bus 消息总线工作原理图

U-Bus 的通信机制将基于 Socket 实现进程间通信，底层将采用 Socket 的工作原理。可以将 U-Bus 的守护进程认为是消息总线的服务器端，进程 A 与进程 B 则为客户端，若要实现进程 A 到进程 B 的通信，需要进程 A 向服务器（守护进程）发送数据，然后服务器接收到数据，最后服务器再将数据发送给 B，数据由进程 B 来接收。

系统中存在多条总线由 U-Bus 总线守护进程管理。U-Bus 守护进程（U-Bus Deamon）是指 U-Bus 在操作系统中提供服务的进程。由系统创建，随操作系统启动而启动，消亡而消亡。其中最主要的总线称作系统总线，系统总线是在 Linux 内核引导的时候就已经装入内存中。

只有内核进程和其他更高权限的进程可以在系统总线上进行写操作，这样就可以保护系统具有较高的安全性，以防止其他恶意进程假冒高优先级进程发送消息到 Linux [16]。普通进程创建会话总线，会话总线可以存在多个。进程间消息的收发的机制是使用会话总线。另外，进程必须注册后才能收到总线中的消息，并且可同时连接到多条总线中。U-Bus 也会像 D-Bus 那样提供匹配器使进程可以有选择性的接收消息，另外运行进程注册回调函数，注册的会掉函数实质上是消息处理函数，当注册进程收到消息后将解析该消息，并调用相应的消息处理函数进行消息的处理 [17]。

如图 1-3 所示，U-Bus 在组成结构上分为四个部分也就是我们要实现的功能点：

（1）接口(Lib-U-Bus)：U-Bus 提供接口函数库，应用程序通过对 API 函数的调用，完成与其他应用程序间的呼叫和交互消息。

（2）守护进程（U-Bus daemon)：总线守护进程可以同时与多个应用程序

相连，并把消息从一个应用程序转到另一个或多个应用程序中。

（3）公共函数、结构体综合库（Lib-U-box）：其中包含了 AVL（完全二叉树）结构、链表等数据结构。U-Bus 实现时需要使用这些资源。

（4） OS/socket：U-Bus 实现需要获得内核消息队列，并且 U-Bus 的底层通讯使用的是 Socket，也需要获取 Socket 资源。

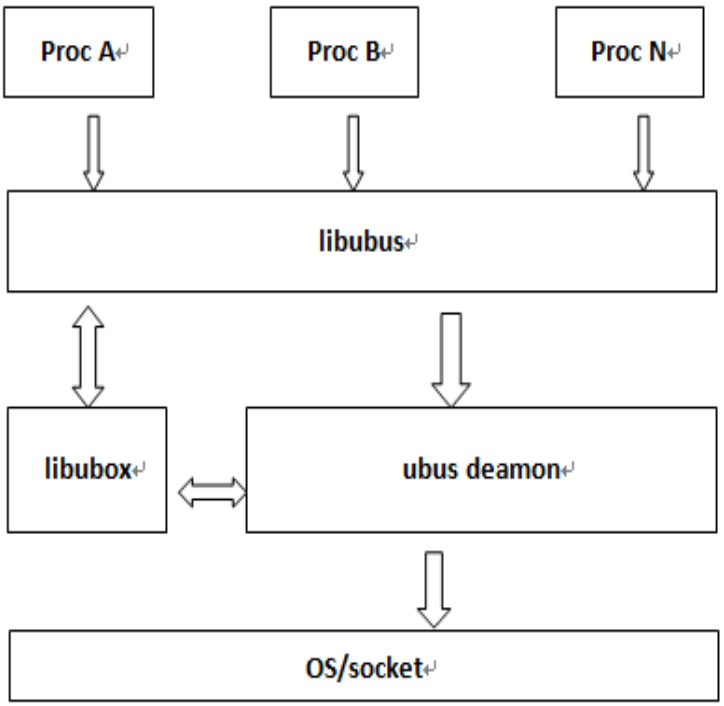

图 1-3　U-Bus 消息总线体系结构图

1.3 本论文的主要工作内容

本课题研究的主要内容是改造、优化蝴蝶路由器所使用的 OpenWrt 操作系统，重新设计蝴蝶路由器的软件系统架构。本课题是来自于蝴蝶路由器研发项目，但是，论文工作是针对优化与改造 OpenWrt 路由器软件系统，研究内容与成果将适用于任何一款基于 OpenWrt 平台的路由器。本课题的研究内容包括以下三个方面：

（1）设计并实现了适用于嵌入式操作系统的消息总线。新设计的消息总线不再臃肿，各模块可以通过消息机制进行通信并且减小系统内存开销。

（2）设计并实现了配置管理模块，弥补原 OpenWrt 没有统一配置管理的缺陷。统一配置管理模块将实现路由器更新配置上的统一和协调。

（3）设计并实现了服务管理模块，服务管理模块将统一管理路由器内部各应用程序的工作；监管各应用程序的工作状态；统一系统服务的注册、注销、启动、关闭。

本论文的章节组织结构如下：

第 1 章：绪论

在结合国内外路由器软件系统文献的基础上，简要介绍本课题的背景与研究目的。重点详细阐述了 OpenWrt 操作系统和国内外各路由器厂商在 Openwrt 平台上的研究与开发现状。最后，说明了本论文的主要研究内容和章节安排。

第 2 章：路由器软件系统的需求分析

在了解系统的业务需求分析的基础上，结合企业的功能需求指标和性能需求指标进行了分析，同时简要介绍了项目中将用到的解决关键技术方法。

第 3 章：路由器软件系统的设计

路由器软件系统的总体设计，重点阐述了本课题所研究的消息总线模块、服务管理模块、配置管理模块的详细设计内容。

第 4 章：路由器软件系统的实现

通过程序流程图和关键代码、函数介绍等方式详细阐述了消息总线模块、服务管理模块、配置管理模块的具体实现。

第 5 章：路由器软件系统功能测试与性能测试

对系统的功能和性能进行测试，并给出评价与分析。

结论：总结了本课题的研究成果，简要介绍了研究方向、后期工作以及不足。

第 2 章 需求分析

本课题将研究如何改造 OpenWrt 以满足蝴蝶路由器的软件系统需求，使其软件系统更加合理，系统运行更有效率，各模块之间的联系更加紧密。原有 OpenWrt 系统中如软件包管理，UCI，WEB GUI 等模块保持不变，大部分模块将进行修改，新增加了配置管理模块、服务管理模块、配置文件管理模块。由于 D-Bus 并不适合蝴蝶路由器软件系统,我们还会重新设计系统中的消息总线。一些新的功能包也将被集成到 OpenWrt 的平台，从而使软件系统更加可靠，功能上容易扩展，以及使用户界面更加友好。新的架构将实现统一配置管理和统一的服务管理。路由器其他功能点例如 WAN、LAN、WiFi、VPN 等模块的实现均通过 OpenWrt 所提供的开源功能软件包来实现，路由器的这些基本功能的实现不在本课题讨论范围之内，只做概述介绍。重点研究的内容包括新设计的消息总线、配置管理模块以及服务管理模块。

2.1 路由器软件系统总体模块划分

如图 2-1 所示，此图所画出的是蝴蝶路由器需要实现的所有模块。整个软件系统的模块结构包括 UI 层、消息转发层、应用程序层、支持库与库文件、OS 层。整个系统的功能依然以软件包的方式实现。各模块间将采用消息机制进行通信。本文在蝴蝶路由器项目中所具体实现的功能模块有以下三个：消息总线模块（M-Bus）、服务管理模块（Service Management），以及配置管理模块（Provisioning Management）。其他模块由 OpenWrt 平台中具体的开源软件包来实现。其中，消息总线层位于消息转发层；配置管理模块和服务管理模块位于应用程序层。将模块按照层次划分如下：

（1）UI 层面将实现 WEB 界面、命令行终端（CLI）、TR069 客户端、SNMP 解析。

（2）消息总线。

（3）应用程序层包括广域网管理程序（WAN）、、TR069 处理程序、SNMP 处理程序、Routing、LAN、WLAN、VLAN、防火墙，以及新增加的服务管理模块和配置管理模块。

（4）支持库：C、C++。

（5）OS 层包括 Linux 内核、驱动程序、Boot Loader（开机启动程序）。

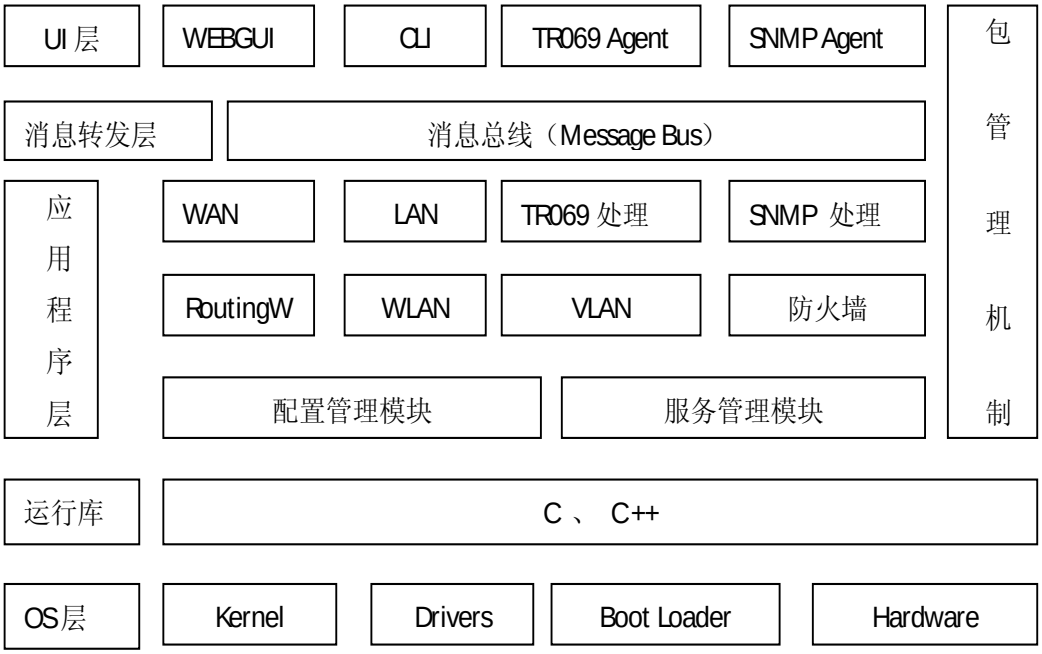

图 2-1　蝴蝶路由器内部模块划分

2.2 路由器软件系统新增模块设计需求

本节对 OpenWrt 平台新增的消息总线模块、服务管理模块、配置管理模块的设计需求进行阐述。

2.2.1 消息总线设计需求

D-Bus 消息总线是面向桌面系统设计，接口丰富，但占用资源较多。重新设计的消息总线将满足占用系统资源少，且可以满足蝴蝶路由器软件系统的消息转发需求。

消息总线（Message Bus，以后简称 M-Bus）模块作为蝴蝶路由器软件系统的基础软件模块，M-Bus 被设计成了一个为路由器操作系统各应用程序提供模块间通信的唯一上层平台。M-Bus 自身被抽象化成一个提供进程间通信方法的函数功能库，负责路由器软件系统各模块间的消息转发和消息广播，实现的方式是向整个系统提供 C 的 API 接口以供其他应用程序调用。M-Bus 底层是使用套接字、信号量、管道等 Linux 基本进程间通信方法进行封装。M-Bus 在消息处理方式是消息的直接转发（Intransit）。消息的直接转发使用命名管道来实现，参与通信的各个进程直接调用 M-Bus 库函数，各个应用程序根据自身注册到消

息总线上的消息处理函数，做出下一步的动作。

另外，我们还要完成系统全局消息的定义。消息总线实现的功能有以下四个方面：

（1）处理事件或应用程序在消息总线上的注册。

（2）处理事件或应用程序在消息总线上的卸载。

（3）消息的发送。

（4）消息的接收。

2.2.2 服务管理模块设计需求

服务管理模块根据配置管理模块的要求，对路由器软件系统各个服务进行维护与管理，包括对各应用程序的进程进行启动、重启和停止等。在 OpenWrt 平台中，一个服务由一个或多个应用程序的进程构成[18]，并注册在服务管理模块中。服务管理模块内部通过所定义的结构体来对各个服务进行管理。定义的数据结构内部记录整个服务的信息包括：进程的 ID 号、进程的名字、进程是否有子进程、路径等其他属性。其次，我们将使用一个双向链表，每一个服务在链表中就是一个节点，节点记录了每个服务的全部信息。我们将所有服务都存到这个双向链表里。另外，服务管理模块还要向配置管理模块提供相应的 API 接口，以供配置管理模块调用：

1）注册 API 函数被调用后，配置管理模块会向服务管理模块发送服务注册请求消息。服务管理收到请求消息后将新的服务加入到服务链表中。

2）注销 API 函数被调用后，配置管理模块会通过消息总线向服务管理模块发送服务注销请求消息。服务管理模块将相关进程停止后，从服务链表中删除。

3）启动 API 函数被调用后，配置管理模块会向服务管理模块发送服务启动请求消息。服务管理模块通过系统调用启动指定的进程。

4）重启 API 函数被调用后，配置管理模块会向服务管理模块发送服务重启请求消息。服务管理模块通过系统调用重新启动指定的进程。

5）停止 API 函数被调用后，配置管理模块会向 Service Manager 模块发送服务停止请求消息。服务管理模块将指定服务下相关的所有进程关掉。

6）当用户需要知道各个服务的运行状态时，通过配置管理模块调用“应用程序状态获取”API 函数，服务管理将通过查询系统目录/proc 下相应 PID 的进程状态，并返回给配置管理模块。

7）当应用程序相对应的进程启动子进程时，向服务管理模块发送特定的告

知消息来注册。服务管理模块内部通过一定的结构管理各个进程及其子进程。

服务管理模块实现以下功能点：

1）统一管理系统各种服务的注册。

2）统一管理系统各种服务的卸载（注销）。

3）统一管理系统各种服务的启动操作。

4）统一管理系统各种服务的关闭（停止）操作。

5）统一管理系统各种服务的重启操作。

6）监控系统中所有服务进程以及其子孙进程。

2.2.3 配置管理模块设计需求

OpenWrt 平台默认使用 UCI（统一配置接口）作为统一的配置访问工具。在 OpenWrt 原有的设计与实现中，各个外部 UI 解析模块都是通过 UCI 的统一调用接口直接操作配置文件。例如：Web GUI 解析模块使用 Lua UCI（也叫 LUCI，用 Lua 脚本语言编写的一个 WEB 页面与 UCI 共同工作的模块）的接口，TR069（广域网管理协议）解析模块、SNMP（简单管理协议）解析模块以及 CLI（路由器命令行）解析模块使用 CLI-UCI 的接口。同时各个服务进程也是通过 Lib-UCI 的接口读取各自的配置文件。

将原有 OpenWrt 平台下的开源代码 UCI（通用配置接口）淘汰，新的软件系统架构将不保留原有的调用 UCI 功能。各外部解析模块直接与新增加的配置管理模块直接进行消息传递，实现模块间的通信。这样做的原因是实现软件整体的低耦合，使各模块在 OpenWrt 平台上更加具备独立性，整个路由器软件系统所占内存更小，运行速度更快。

在蝴蝶路由器软件系统中，由配置管理模块（Provisioning Management）统一管理路由器配置和操作维护。这种统一管理配置和操作的架构设计有利于扩展配置和操作的接口，以及集成新的服务。其中包括以下几个方面：

1）系统重启恢复出厂设置。

2）对所有临时配置文件的修改操作。

3）对永久性存储配置文件的读写操作。

4）向服务管理模块发起服务的注册、注销（卸载）、启动、停止、重启操作命令。

2.3 关键问题与解决方案

（1）课题要求蝴蝶路由器的软件系统所占 ROM 存储要求小于 8MB。OpenWrt 是用软件功能包来实现各个模块的功能，软件功能包一个一个加上去。但是 OpenWrt 系统还包含了我们所要完成系统需求以外的软件包，在不筛选 OpenWrt 系统源代码而选择直接编译的时候，所生成的系统大小约在 2.8GB 左右，这就要求我们首先明确每个系统功能是通过哪些个软件包可以实现的，然后筛选出我们不需要的软件功能包。筛选的工作量十分大。而且，我们在设计新模块的时候也要考虑代码量，检查数据结构的合理性，尽量的使代码简洁，使新设计的三个模块所占内存总大小要尽量的小于 650KB。

（2）服务管理模块拥有监管各进程运行状态的功能，但是被监管的各个进程还会派生出各自的子进程，各子进程有可能继续派生出孙子进程。服务管理模块只能监管第一层次上的进程（父进程）。这就需要父进程在派生出子进程的时候收集子进程的参数，将子进程的相关信息全部收集。同理，子进程在派生孙子进程的时候也需要保留孙子进程的信息，这个操作是递归的。当一个进程派生子进程时就要上报，将新派生的子进程 ID 号上报给服务管理模块。因此，服务管理模块仅通过父进程掌握的信息就拿到了其子孙进程的全部信息。

2.4 本章小结

本章主要是对改造后的路由器软件系统需求进行分析，在原有 OpenWrt 平台上，通过增加配置管理模块实现路由器具有对配置处理操作统一管理的功能；增加服务管理模块弥补原有 OpenWrt 系统没有统一服务管理这一缺陷；重新设计消息总线，使路由系统模块间通信效率更高，系统开销更小。另外，本章详细阐述了本课题研究的消息总线、服务管理模块、配置管理模块的设计需求，通过设计并增加以上三个模块以达到优化原有 OpenWrt 系统的目的。并对存在的困难问题与解决方案进行了阐述。

第 3 章 路由器软件系统的优化设计

本章介绍蝴蝶路由器软件系统优化设计部分。首先从总体架构的角度将路由器软件系统分为各个模块，并对路由器各模块的工作原理、路由器软件系统中各应用程序的启动顺序进行设计。其次，将详细阐述在 OpenWrt 平台上重新设计的消息总线模块、服务管理模块、配置管理模块的详细设计过程。

3.1 软件系统总体架构设计

重点设计与优化的内容包括新设计的消息总线模块、配置管理模块（Provisioning Management）以及服务管理模块（Service Management）。其中，将原有 OpenWrt 平台下的开源代码 UCI（通用配置接口）淘汰，新的软件系统架构将不保留原有的调用 UCI 功能。各外部解析模块直接与新增加的配置管理模块直接进行消息传递，实现模块间的通信。这样做的原因是实现软件整体的低耦合，使各模块在 OpenWrt 平台上更加具备独立性，整个路由器软件系统所占内存更小，运行速度更快。改造后的路由器软件系统架构如图 3-1 所示。

蝴蝶路由器软件系统基于 OpenWrt 平台，根据产品统一管理配置和操作的需求，系统设计在 OpenWrt 平台基础之上进行必要的架构改造和扩展。架构图的层次从上至下顺序号依次增加，第二层是各外部应用程序所对应的解析模块，各个解析模块负责将来自外部的数据进行解析，然后通过消息总线将数据提供给配置管理模块。各解析模块是整个路由器系统的最上层外部接口，也是路由器系统与外部应用程序之间联系的桥梁。

在蝴蝶路由器软件系统的架构设计中，增加了配置管理模块（Provisioning Management）和服务管理模块（Service Management），以及统一的消息总线模块（Message Bus）。配置管理模块用于统一管理系统的配置和操作。服务管理模块用于管理系统中所有服务，包括启动服务、重启服务、关闭服务相关的进程，以及监控它们的运行状态等。消息总线模块为其他系统模块以及服务进程提供消息转发服务。同时为了将各个解析模块的配置与操作都通过消息定向到配置管理模块进行统一管理。

架构图的第三层就是我们上述所说的是新设计的配置管理（Provisioning Management）模块，是整个软件系统的核心，用于接收各解析模块发来的消息，配置管理模块将对所获得的数据进行分析，然后根据不同的情况与选择对配置文件进行读写或修改操作，以及通过消息向服务管理模块发送相应的服务修改

命令，如启动或关闭等。架构图的第四层就是我们上述所说的新增加的服务管理模块（Service Management），目的是为了实现路由器内部各服务的统一管理。

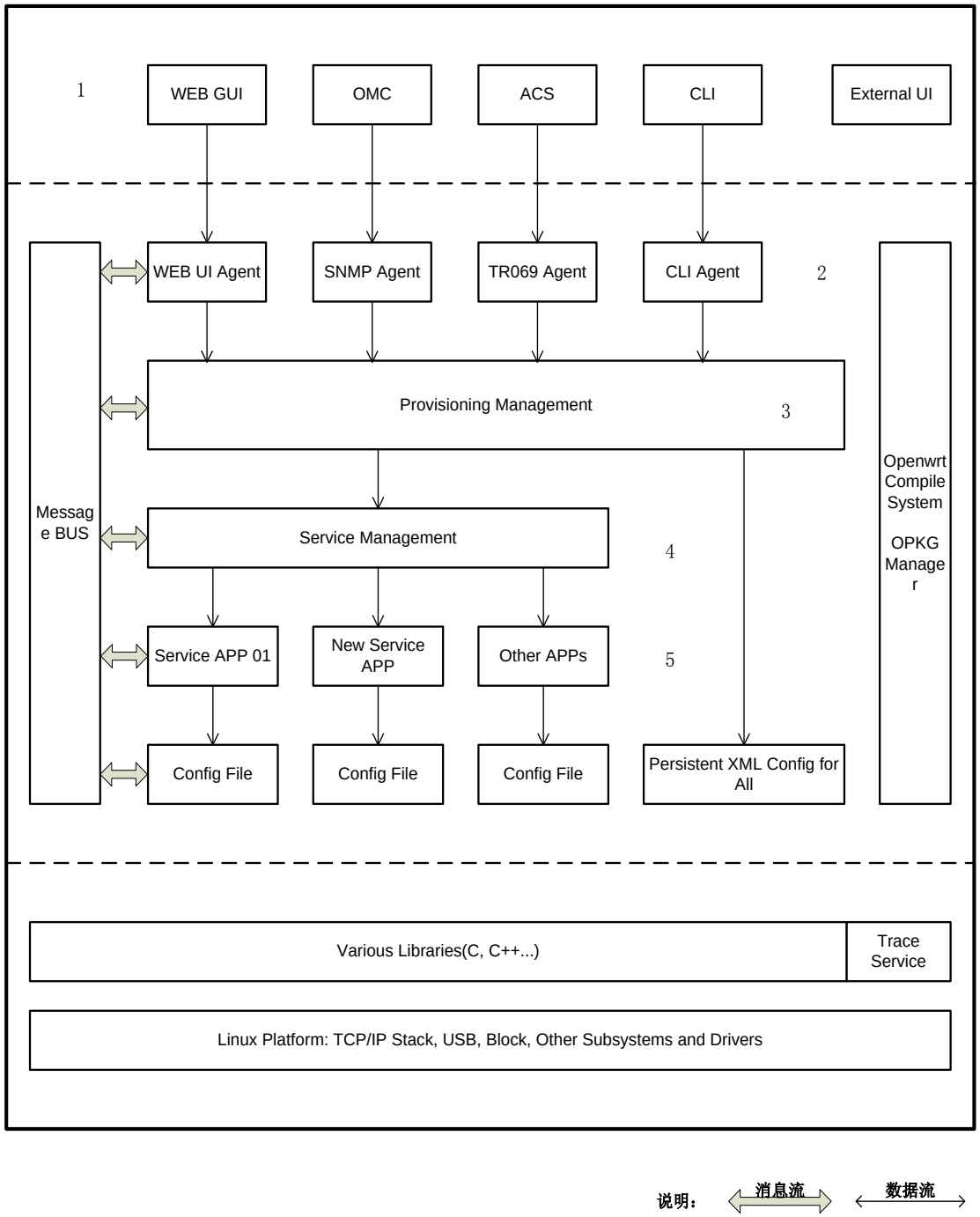

图 3-1　改造后软件系统架构图

整个软件系统将使用新设计的消息总线，命名为 M-Bus，M-Bus 仍然扮演着消息总线的角色，各个进程可以通过 M-Bus 来进行通信。

3.2 系统应用程序启动顺序的设计

图 3-2 描述了路由器软件系统各应用程序的启动顺序：

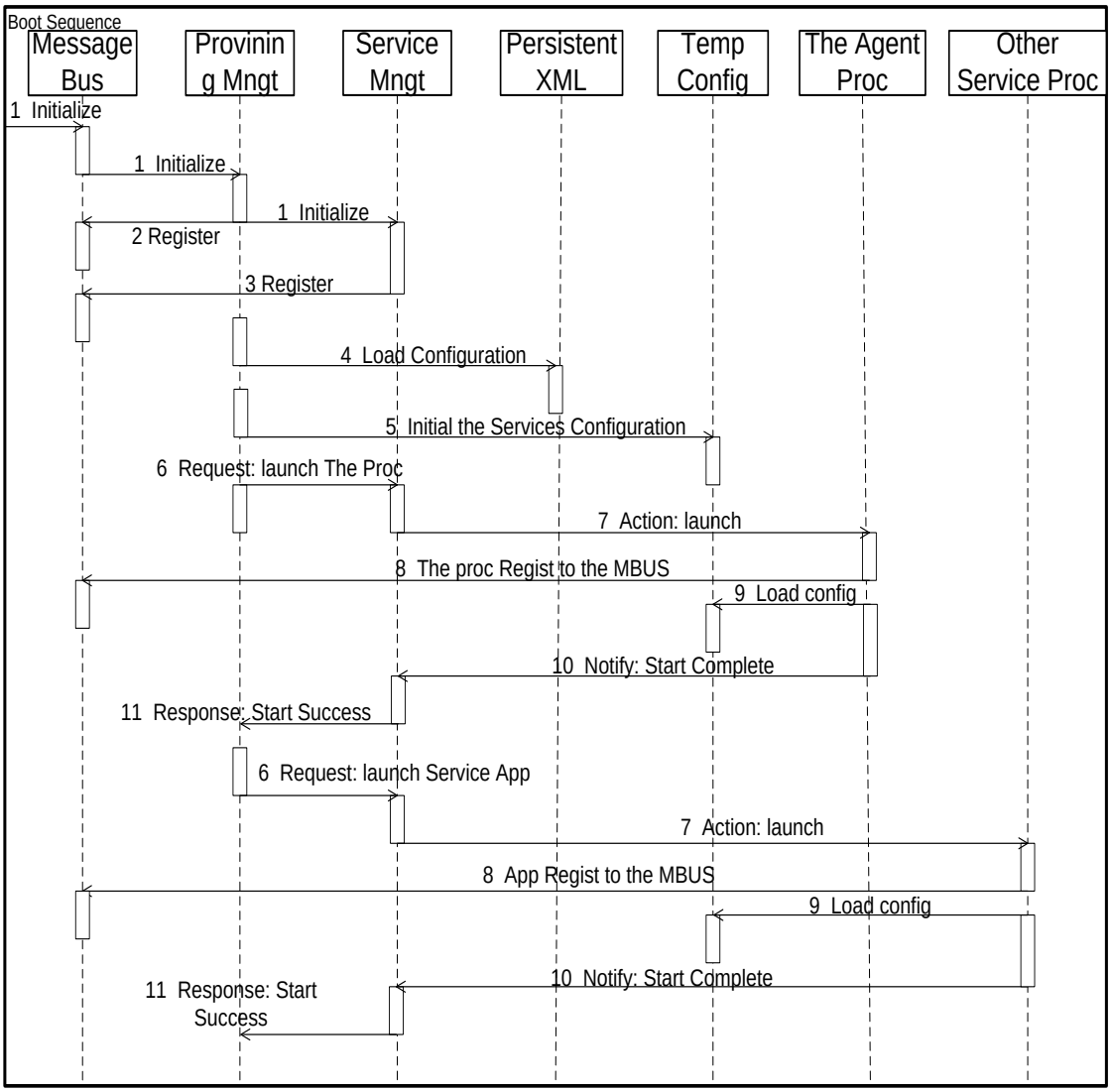

图 3-2　软件系统各模块启动时序图

首先，消息总线（Message Bus）模块作为系统转发消息的基础模块需要最先初始化，M-Bus 的守护进程首先运行，在/var/mbus 目录下生成日志文件，后期从消息总线上注册、卸载的应用程序的记录均保存在该目录下。

配置管理模块（Provisioning Management）作为系统的核心模块也要在开始位置被初始化，并且将自身注册到消息总线上。然后配置管理模块需要初始化服务管理模块（Service Management），并读取永久性存储配置文件，通过各

个服务注册的 handler 函数，生成相应的临时配置文件，依次启动各个 UI 解析模

块以及各个服务模块，各 UI 解析模块与各服务模块启动后首先注册到消息总线上，然后读取临时配置文件，生成自身相应的配置，启动成功后将告知服务管理模块。最后，服务管理模块反馈给配置管理模块相应的结果。当所有的 UI 解析模块与应用程序都加载运行之后，路由器开始工作。

3.3 消息总线模块的设计

消息总线模块的设计将从消息总线总体设计、全局消息定义、数据结构的设计三个方面进行阐述。

3.3.1 消息总线模块总体设计

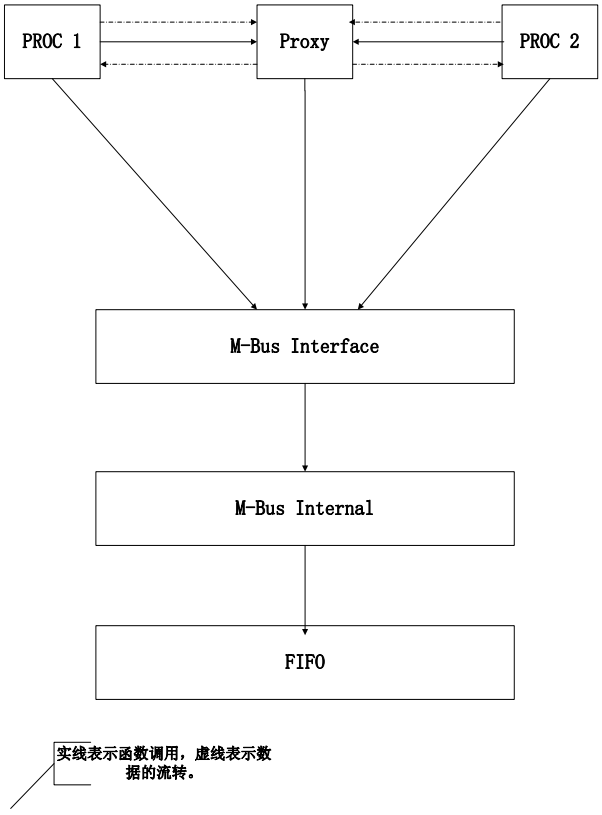

图 3-3　消息总线模块结构图

消息总线（Message Bus，以后简称 M-Bus）模块作为蝴蝶路由器软件系统的基础软件模块，M-Bus 被设计成了一个为路由器操作系统各应用程序提供模块间通信的唯一上层平台。M-Bus 自身被抽象化成一个提供进程间通信方法的

函数功能库，负责路由器软件系统各模块间的消息转发和消息广播，实现的方式是向整个系统提供 C 的 API 接口以供其他应用程序调用。M-Bus 底层是使用套接字、信号量、管道等 Linux 基本进程间通信方法进行封装。M-Bus 在消息处理方式是消息的直接转发。消息的直接转发使用命名管道来实现，参与通信的各个进程直接调用 M-Bus 库函数，各个应用程序根据自身注册到消息总线上的消息处理函数，做出下一步的动作。图 3-3 描述了消息总线的模块组成。

M-Bus 包括以下三个子模块：

（1）M-Bus Interface: M-Bus 的接口集合，包括消息的发送、消息的接收、消息发送者与接收者的登记等一系列 M-BUS 能够提供的 API 函数。

（2）M-Bus Proxy: M-Bus 的守护进程。

（3）M-Bus Internal：M-Bus 内部工作处理，为上层 API 函数提供基础。

路由器的各应用程序通过调用 M-Bus 的 API 函数来使用 M-Bus 的功能。M-Bus 提供了本地资源初始化、销毁本地资源、注册、卸载、发送消息、接收消息、登记消息处理等 API 函数，表 3-1 列出了 M-Bus 的 API 函数。在第四章中，将说明消息总线的具体实现。

表 3-1 M-Bus 接口函数

编号	M-Bus 接口	功能说明
1	int mbus_init	M-Bus 资源初始化，生成 M-Bus 日志目录
2	mbus_obj_register	进程注册到 M-Bus
3	mbus_obj_unregister	进程从 M-Bus 上卸载
4	mbus_receive	收消息
5	mbus_send	发消息
6	mbus_name_locate	获取文件操作句柄
7	mbus_reply	收到消息后回复发送者（用于同步操作）
8	mbus_register_event_map	注册消息处理函数
9	mbus_msg_loop	带循环的收消息（mbus_receive 函数封装）
10	mbus_terminate	销毁本地资源

消息总线中所定义的消息，是进程间传递数据的载体，消息的定义遵循以下原则：

（1）每个消息都有自身的名字，消息的名字表示要发送的消息是什么命令[19]。

（2）消息的名字在系统中是唯一的[20]。

（3）系统能处理消息的种类的能力是有限的，表 3-2 列出了蝴蝶路由器软件系统中定义的全部消息名字，全局消息以枚举类型被定义。各个模块收到消息后会跟据消息的名字执行相应的处理函数，例如服务管理模块收到 SYS_MSG_START_SERVICE 消息，服务管理模块就会执行启动某服务的函数。

（4）消息具有统一定义的数据结构，包括消息头、携带数据、消息上下文（Context）[21]。消息的名字（也可称为消息的类型）作为消息头中的一个数据域的形式存在。关于消息数据结构的定义参见下一章节。

表 3-2 全局消息定义

编号	消息名字	说明
1	SYS_MSG_RESERVED	SYS_MSG_RESERVED =0x0000
2	SYS_MSG_APP_LAUNCHED,	用于服务管理模块确定某个特定服务已经成功启动
3	SYS_MSG_REBOOT_SYSTEM	配置管理模块重启操作系统
4	SYS_MSG_GET_PARAM_VAL	读取特定参数的值
5	SYS_MSG_SET_PARAM_VAL	设置某个特定参数的值
6	SYS_MSG_GET_PARAM_NAME	读取特定参数的名字
7	SYS_MSG_GET_PARAM_ATTR	读取特定参数属性
8	SYS_MSG_SET_PARAM_ATTR	设置某个特定参数属性
9	SYS_MSG_UPLOAD	将路由器配置信息上传给服务商
10	SYS_MSG_DOWNLOAD	下载
11	SYS_MSG_ADD_OBJ	增加 Object（XML 格式的永久性存储文件中存在的一个属性）
12	SYS_MSG_DEL_OBJ	删除 Object
13	SYS_MSG_MODIFY_OBJ_ATTR	修改 Object 的属性
14	SYS_MSG_ACS_CONFIG_CHANGED	ACS 服务端的配置文件改变
15	SYS_MSG_TR69_ACTIVE_NOTIFICATION	当一个参数值改变，则发送至 TR069 解析段告知
16	SYS_MSG_DHCPC_STATE_CHANGED	从 DHCP 服务发送的状态改变告知
17	SYS_MSG_PPPOE_STATE_CHANGED	从 PPPOE 服务发送的状态改变告知

表 3-2（续表）

编号	消息名字	说明
18	SYS_MSG_DHCP6C_STATE_CHANGED	从 dhcpv6 服务发送的状态改变告知
19	SYS_MSG_DHCPD_RELOAD	DHCPD 配置更新后重新加载配置
20	SYS_MSG_DHCPD_HOST_INFO	DHCPD 通知 LAN 增加与删除
21	SYS_MSG_WAN_LINK_UP	WAN 连接 UP（成功得到 IP）
22	SYS_MSG_WAN_LINK_DOWN	WAN 连接 DOWN
23	SYS_MSG_ETH_LINK_UP	以太网连接成功
24	SYS_MSG_ETH_LINK_DOWN	以太网连接关闭
25	SYS_MSG_USB_LINK_UP	USB 连接成功
26	SYS_MSG_USB_LINK_DOWN	USB 连接失败
27	SYS_MSG_DNSPROXY_RELOAD	DNS 守护进程重新加载配置
28	SYS_MSG_DNSPROXY_DUMP_STATUS	告知 DNS 守护进程清空自身配置
29	SYS_MSG_DNSPROXY_DUMP_STATS	告知 DNS 清空全部存在的状态
30	SYS_MSG_SNMPD_CONFIG_CHANGED	SNMPD 配置改变
31	SYS_MSG_STORAGE_ADD_PHY_MEDIA SYS_MSG_STORAGE_REMOVE_PHY_MEDIA SYS_MSG_STORAGE_ADD_LOGIC_VLM SYS_MSG_STORAGE_REMOVE_LOGIC_VLM	四种存储管理消息
32	SYS_MSG_START_SERVICE	请求服务管理模块启动某个服务
33	SYS_MSG_STOP_SERVICE	请求服务管理模块停止某个服务
34	SYS_MSG_RESTART_SERVICE	请求服务管理模块重启某个服务
35	SYS_MSG_SRVMNGR_APP_EXIT	请求服务管理模块停止某应用程序
36	SYS_MSG_IS_SERVICE_RUNNING	请求服务管理模块检查某个服务是否在运行
37	SYS_MSG_SRVMNGR_CHILD_CTEATE	告知服务管理模块该进程将创建子进程
38	SYS_MSG_SRVMNGR_APP_LAUNCHED	启动某个应用程序
39	SYS_MSG_SET_LOG_LEVEL	请求 app 设置其自身 LOG_LEVEL
40	SYS_MSG_SET_LOG_DESTINATION	请求 app 设置其自身 LOG 地址

表 3-2（续表）

编号	消息名字	说明
41	SYS_MSG_MEM_DUMP_STATS	请求服务管理模块清空 app 的状态
42	SYS_MSG_MEM_DUMP_TRACEALL	请求服务管理模块清空 app
43	SYS_MSG_GET_WAN_LINK_STATUS	请求读取当前 WAN 连接的状态
44	SYS_MSG_GET_LAN_LINK_STATUS	请求当前 LAN 连接的状态
45	SYS_MSG_REQUEST_FOR_PPP_CHANGE	请求断开 PPP
46	SYS_MSG_SRVMNGR_REG_APP SYS_MSG_SRVMNGR_UNREG_APP	通知服务管理模块注册/卸载某应用程序
47	SYS_MSG_SRVMNGR_SET_PRIORITY	通知服务管理模块设置优先权

3.3.2 消息总线数据结构的设计

消息的自身是数据传递的载体并且消息具有相应的结构。消息结构组织分为两类：一类是各模块之间通信的消息结构；另一类是各模块本地维护的消息结构[22-23]。

其中，消息头被定义成各模块间通信的唯一结构，各模块间的通信是通过解析消息头来提取数据，从而实现进程间的通信。而各模块本地维护的消息结构称之为 Message Context（消息上下文），每个模块都会有自身的消息上下文，由各个模块自己组织与管理，与外界隔离。

图 3-4 描述了消息头的数据结构：

MBUS_MSG_HDR

发送者名字 **char sender[32]**	**当前进程ID** **pid_t pid**	**消息名字** **int msg_type**	**同步标志位** **short sync**	**消息长度** **int msg_len**	**消息体** **数据起始地址** **Char* data**

图 3-4　消息头数据结构示意图

消息头中包含以下定义内容：

（1）消息的发送者：定义该消息是由哪个模块发送的，路由器所有模块的名称均用宏定义。

（2）当前进程 PID：该消息的发送进程的 PID（进程号）。

（3）消息的名字：该消息发的是什么指令。

（4）消息的同步：当接收进程收到消息后需要做反馈操作，回复发送进程

进行收到确认。如果不做同步操作则不需要回复。

（5）数据长度：消息所携带的数据长度。

（6）携带数据起始位：所携带数据的起始位地址。起始位地址加上所携带数据的长度就可以表示该消息携带的所有数据，即消息的消息体（Message Body）。

我们称各模块自身维护的消息为本地消息，描述本地消息的数据结构称为消息上下文（Message Context），路由器软件系统中每个模块（每个应用程序）自身只能存在一个消息上下文。设计消息上下文的原因在于是想把使用消息总线的所有数据与操作方式都组织到一起，然后封装成统一的结构来进行描述。每个应用程序注册到消息总线上的时候，都会生成自身的消息上下文。图 3-5 描述了消息上下文的数据结构定义。

MBUS_CONTEXT

应用程序名 char my_name[32]	当前进程ID Pid_t cur_pid	文件操作句柄 Int recv_fd	退出函数 pf_user_exit user_exit	消息处理函数 MBUS_EVENT_MAP *event_map	默认消息处理函数 mbus_event_handler default_handler	消息头 MBUS_MSG_HDR msg_hdr

图 3-5 消息上下文数据结构示意图

消息上下文中包括了以下内容：

（1）注册到消息总线上的应用程序自身名字。名字由字符串表示，系统中所有的应用程序名字均使用宏进行定义。表 3-3 描述了所有名字的宏定义。

表 3-3 全局应用程序宏定义

编号	宏	应用程序名字对应的字符串
1	MBUS_PROXY_NAME	"mbus-proxy"
2	APP_PROVISION_MNGT_NAME	"provisioning"
3	APP_SERVICE_MNGR_NAME	"serv-mngr"
4	APP_TR069C_NAME	"tr069c"
5	APP_HTTPD_NAME	"httpd"
6	APP_SNMPD_NAME	"snmpd"
7	APP_CONSOLED_NAME	"consoled"
8	APP_TELNETD_NAME	"telnetd"
9	APP_SSHD_NAME	"sshd"
10	APP_UPNP_NAME	"upnp"
11	APP_WAN_MNGR_NAME	"wan-mngr"

表 3-3（续表）

编号	宏	应用程序名字对应的字符串
12	APP_WLAN_MNGR_NAME	"wlan-mngr"
13	APP_LAN_MNGR_NAME	"lan-mngr"
14	APP_PPP_NAME	"ppp"
15	APP_DSL_DIAG_NAME	"dsl-diag"
16	APP_VPN_MNGR_NAME	"vpn-mngr"
17	APP_VLAN_MNGR_NAME	"vlan-mngr"
18	APP_FIREWALL_MNGR_NAME	"firewall-mngr"
19	APP_DHCPC_NAME	"dhcpc"
20	APP_DHCPD_NAME	"dhcpd"
21	APP_DHCP6C_NAME	"dhcp6c"
22	APP_DHCP6S_NAME	"dhcp6s"
23	APP_FTPD_NAME	"ftpd"
24	APP_TFTPD_NAME	"tftpd"
25	APP_TFTP_NAME	"tftp"
26	APP_DNS_PROBE_NAME	"dns-probe"
27	APP_DDNSD_NAME	"ddnsd"
28	APP_DNS_PROXY_NAME	"dns-proxy"
29	APP_SYSLOGD_NAME	"syslogd"
30	APP_KLOGD_NAME	"klogd"
31	APP_RIPD_NAME	"ripd"
32	APP_IGMP_NAME	"igmp"
33	APP_HOTPLUG_NAME	"hotplug"
34	APP_SAMBA_NAME	"samba"
35	APP_L2TPD_NAME	"l2tpd"
36	APP_PPTPD_NAME	"pptpd"
37	APP_UCIMSG_NAME	"ucimsg"
38	APP_DNSMASQ_NAME	"dnsmasq"

（2）当前注册到消息总线上的进程 ID。进程 ID 用于表示消息总线使用者的身份。

（3）当前接收消息的文件描述符。当一个进程新注册到消息总线上时，该文件描述符设置为-1。当该进程参与消息的发送或接收时，该文件描述符表示文件操作句柄。

（4）进程退出函数指针，typedef void (*pf_user_exit)(void)。当一个已在消息总线上注册过的进程想要从消息总线上卸载时，调用自身定义的退出函数

实现退出。

（5）消息处理函数指针。当应用程序收到消息时解析该消息，根据解析到的消息名字调用相对应的消息处理函数。当应用程序向消息总线上注册时，必须注册对应消息的处理函数。

（6）默认消息处理函数指针。

（7）消息头。

3.4 服务管理模块的设计

3.4.1 服务管理模块总体设计

服务管理模块根据配置管理模块的要求，对路由器软件系统各个服务进行维护与管理，包括对各应用程序的进程进行启动、重启和停止等。在 OpenWrt 平台中，一个服务由一个或多个应用程序的进程构成，并注册在服务管理模块中。服务管理模块内部通过所定义的结构体来对各个服务进行管理。定义的数据结构内部记录整个服务的信息包括：进程的 ID 号、进程的名字、进程是否有子进程、路径等其他属性。其次，我们将使用一个双向链表，每一个服务在链表中就是一个节点，节点记录了每个服务的全部信息。我们将所有服务都存到这个双向链表里。另外，服务管理模块还要向配置管理模块提供相应的 API 接口，以供配置管理模块调用：

（1）注册 API 函数被调用后，配置管理模块会向服务管理模块发送服务注册请求消息。服务管理收到请求消息后将新的服务加入到服务链表中。

（2）注销 API 函数被调用后，配置管理模块会通过消息总线向服务管理模块发送服务注销请求消息。服务管理模块将相关进程停止后，从服务链表中删除。

（3）启动 API 函数被调用后，配置管理模块会向服务管理模块发送服务启动请求消息。服务管理模块通过系统调用启动指定的进程。

（4）重启 API 函数被调用后，配置管理模块会向服务管理模块发送服务重启请求消息。服务管理模块通过系统调用重新启动指定的进程。

（5）停止 API 函数被调用后，配置管理模块会向 Service Manager 模块发送服务停止请求消息。服务管理模块将指定服务下相关的所有进程关掉。

（6）当用户需要知道各个服务的运行状态时，通过配置管理模块调用“应用程序状态获取”API 函数，服务管理将通过查询系统目录/proc 下相应 PID 的进程状态，并返回给配置管理模块。

（7）当应用程序相对应的进程启动子进程时，向服务管理模块发送特定的告知消息来注册。服务管理模块内部通过一定的结构管理各个进程及其子进程。

服务管理模块实现以下功能点：

（1）统一管理系统各种服务的注册。

（2）统一管理系统各种服务的卸载（注销）。

（3）统一管理系统各种服务的启动操作。

（4）统一管理系统各种服务的关闭（停止）操作。

（5）统一管理系统各种服务的重启操作。

（6）监控系统中所有服务进程以及其子孙进程。

图 3-6 描述了服务管理模块的整体组成结构：

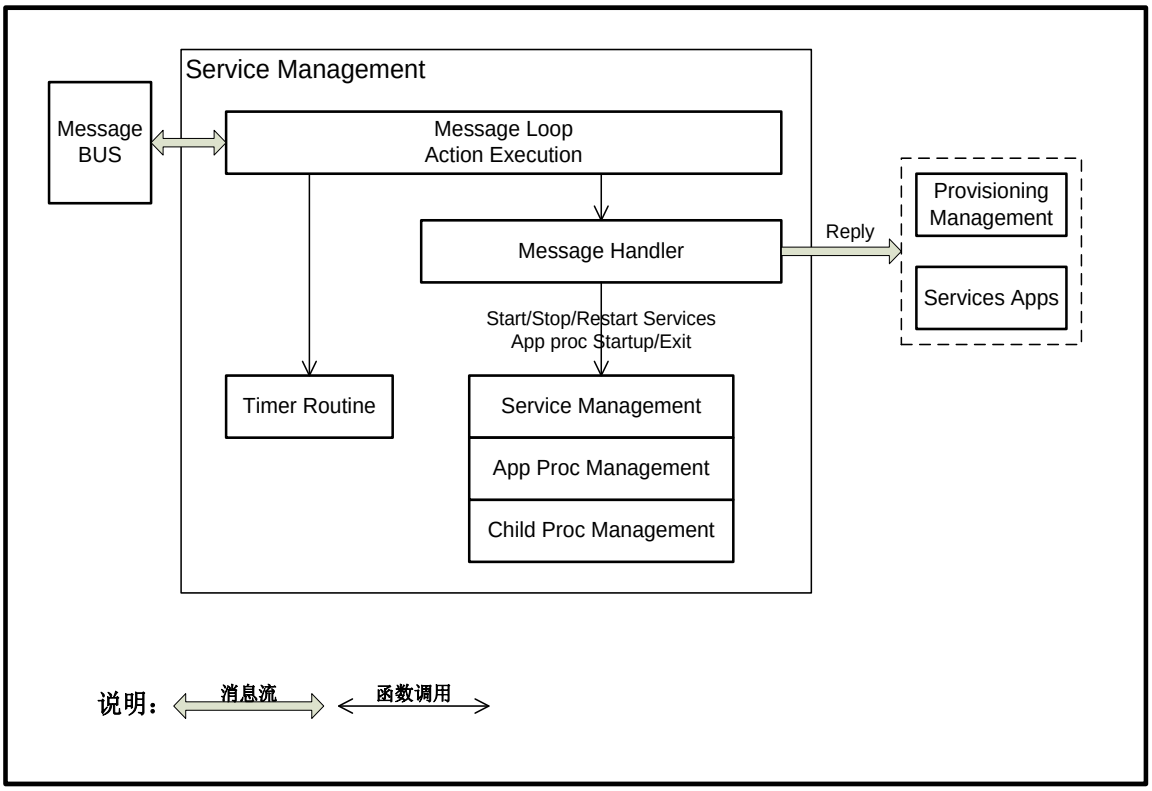

图 3-6 服务管理模块结构示意图

当路由器软件系统启动 Service Management 后，Service Management 将进入无限循环一直监听消息（Message Loop）。当收到消息后，将调用与该消息对应的消息处理函数（Message Handler），从而对本地数据结构进行操作。另外，服务管理模块也设计了定时处理（Timer Routine），用来定时遍历系统根目录/proc 下的文件，及时更新各服务对应进程的信息。

3.4.2 服务管理模块数据结构的设计

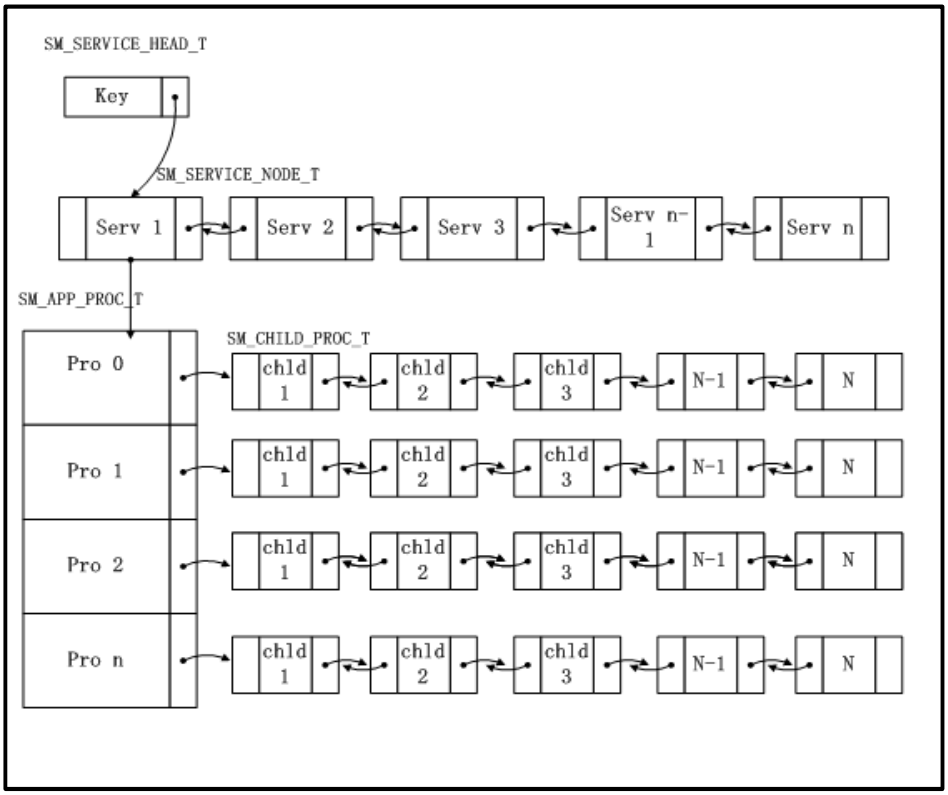

图 3-7 服务管理模块数据结构示意图

在蝴蝶路由器软件系统中，一个服务（Service）由若干个应用程序主进程构成（存在只有一个应用程序主进程构成一个服务）[24-25]，每个应用程序是由其自身主进程和其派生出的若干子进程所构成[26]。系统中的所有服务，由双向循环链表组织在一起，并用一个头指针来指向服务链表的头部。每个服务节点下，都包含了一个能容纳 8 个应用程序主进程的数组，数组中的每个元素，存放的是该服务对应的应用程序主进程。主进程有其自身定义的结构，主进程的结构中包含了其派生出子进程的头指针。子进程由双向链表组织在一起。

各结构均使用双向链表的形式实现更有利于对各个节点的管理，双向链表在节点的插入与删除操作上效率更高，例如某个服务增加了一个进程，则创建一个进程对应结构的内存空间，将该进程的信息存入此节点，直接插入到相对应的双向链表中即可。当某个子进程终止，根据子进程标识号直接定位存放该子进程的节点将其删除。图 3-7 描述了服务管理模块数据结构整体示意图。

服务管理模块使用结构体 SM_APP_PROC_T 和 SM_CHILD_PROC_T 来管理某个服务下的各个进程。所有的应用程序进程将以这种形式存储在 Service

Table（服务列表）中。父进程下的子进程将以链表形式进行管理。应用程序进程的结构体簇定义如表 3-4 与 3-5 所示：

表 3-4 SM_CHILD_PROC_T结构的定义

类型	数据域	说明
int	iPPid	子进程产生时，由系统分配的 PPID，PPID 父进程 ID
int	iPid	子进程产生时，由系统分配的 PID，子进程自身 ID
int	iRunStateFlag	标识子进程状态标志位:0：非运行，非 0：运行中
char*	acAppName	该子进程隶属于的应用程序名
SM_CHILD_PROC_T*	pstProcNext	指向下一个子进程节点的指针
SM_CHILD_PROC_T*	pstProcPri	指向上一个子进程节点的指针

表 3-5 SM_APP_PROC_T结构的定义

类型	数据域	说明
int	iPid	当前进程 ID
int	iAppID	与配置管理模块定义的 App ID 一致
int	iRunStateFlag	标识进程状态标志位:0：非运行，非 0：运行中
char*	acAppName[]	该进程隶属于的应用程序名
char*	pcPath	进程路径
unsigned short	ulStackSize	进程运行时栈大小
unsigned short	usPriority	进程优先级
unsigned long	ulChildNum	当前进程含有子进程个数
SM_CHILD_PROC_T*	pstChildProc	子进程的头结点地址

服务列表将用双向链表的形式实现。结构 SM_SERVICE_HEAD_T 用于声明服务列表的全局变量,结构体 SM_SERVICE_NODE_T 包含服务以及服务下各应用程序的具体信息。服务列表结构体簇定义如表 3-6 与 3-7 所示：

表 3-6 SM_SERVICE_NODE_T结构的定义

类型	数据域	说明
char	iPid	当前进程 ID
int	iAppID	与配置管理模块定义的 App ID 一致
int	iRunStateFlag	标识进程状态标志位:0：非运行，非 0：
SM_APP_PROC_T [8]	astAppProc	应用程序进程数组，长度 8
int	iProcNum	该服务有效的进程数
SM_SERVICE_NODE_T *	pNextServ	指向下一个服务节点的指针
SM_SERVICE_NODE_T *	pPriorServ	指向上一个服务节点的指针

表 3-7 SM_SERVICE_HEAD_T结构的定义

类型	数据域	说明
SM_SERVICE_NODE_T *	pServiceNode	指向一个服务节点的指针
unsigned long	ulServiceNum	该服务双向链表节点总个数

3.4.3 定时处理

当服务管理模块从消息总线上解析到的数据大小为 0 时，将调用定时处理函数。采用定时处理，遍历软件系统根目录下的/proc 目录，根据获取的进程名与本地比较，进行更新 PPID（父进程 PID）、子程序添加等操作。

3.5 配置管理模块的设计

配置管理模块的设计将从该模块的总体设计、配置处理、数据模型管理、适配子模块四个方面进行阐述。

3.5.1 配置管理模块总体设计

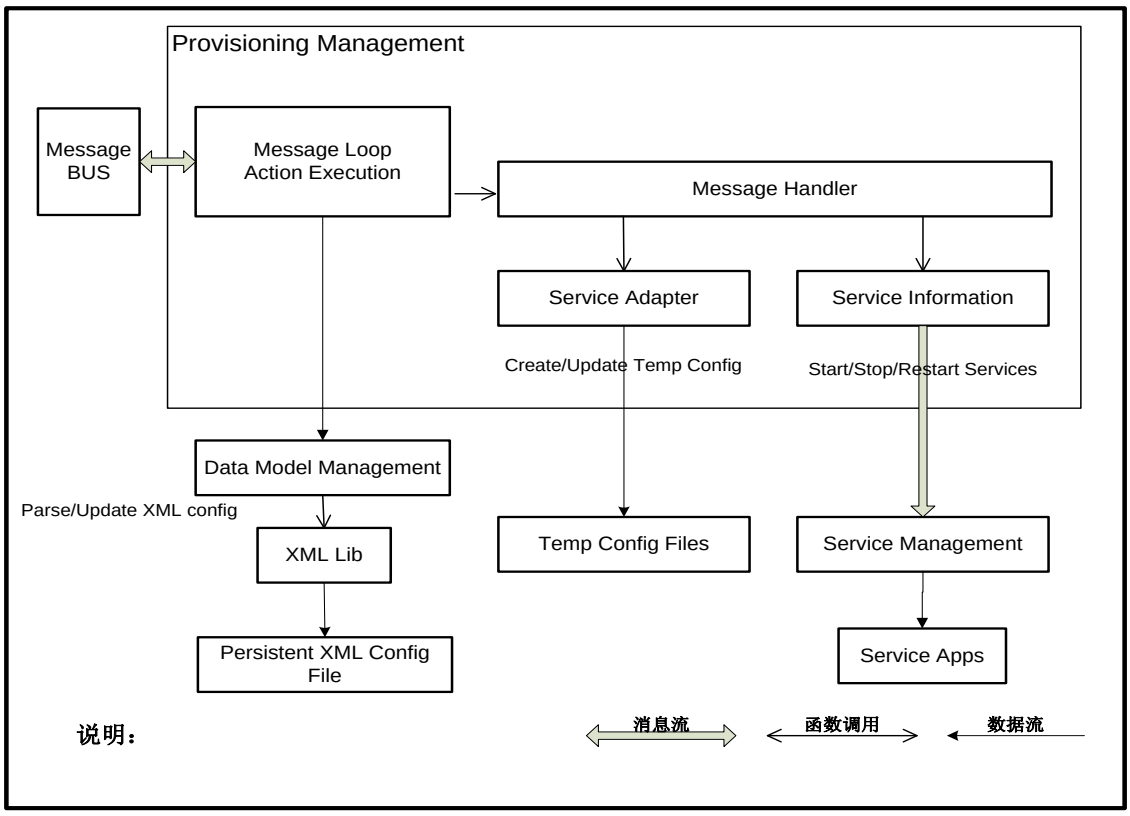

图 3-8 配置管理模块结构示意图

图 3-8 描述了配置管理模块的内部功能划分。蝴蝶路由器软件系统中所有

的配置数据都保存在 XML 格式的永久性配置文件中。各个服务对应的配置文件由配置管理模块通过各个服务注册处理函数的方式，进行统一修改更新，因此称它们为临时配置文件，临时文件位于内存中。各个 UI 解析模块发送配置读写，系统重启等请求消息给配置管理模块，配置管理模块解析这些消息请求完成相应的操作。比如针对配置读写请求：直接读写相应的临时配置文件。然后通知服务管理模块启动、重启、停止相应的服务，并等待服务管理模块返回相应的操作结果。如果配置操作成功，则更新永久性存储配置文件，永久性存储文件位于磁盘当中。如果配置失败，则重新加载永久性存储配置文件中的相应配置数据，并以此数据恢复相应的临时配置文件，然后重新启动相应的服务。

配置管理模块通过 Message Handling（消息处理）子模块向消息总线注册一组可以接收并处理的消息及其 handler 处理函数。配置管理模块可以处理的消息包括：

（1）获取、设置参数值。

（2）获取参数名字。

（3）获取/设置参数的属性。

（4）增加、删除 object。

（5）获取、修改 object 属性。

（6）系统重启。

（7）恢复出厂设置。

（8）服务启动成功或失败的 ACK（命令确认字），上传或下载等。

这些消息和对应处理函数的映射关系如 pm_event_methods 所定义：

```
MBUS_EVENT_MAP provisioning_event_map[] =
{
    {SYS_MSG_GET_PARAM_VALUE,          pm_hdl_get_param_val },
    {SYS_MSG_SET_PARAM_VALUE,          pm_hdl_set_param_val },
    {SYS_MSG_GET_PARAM_NAMES,          pm_hdl_get_param_name},
    {SYS_MSG_GET_PARAM_ATTRIBUTES,     pm_hdl_get_param_attr },
    {SYS_MSG_SET_PARAM_ATTRIBUTES,     pm_hdl_set_param_attr },
    {SYS_MSG_ADD_OBJ,                  pm_hdl_add_obj        },
    {SYS_MSG_DEL_OBJ,                  pm_hdl_del_obj        },
    {SYS_MSG_UPLOAD,                   pm_hdl_upload         },
    {SYS_MSG_DOWNLOAD,                 pm_hdl_download       },
    {SYS_MSG_REBOOT,                   pm_hdl_reboot         },
    {SYS_MSG_FACTORY_RESET,            pm_hdl_factory_reset  },
    {SYS_MSG_GET_OBJ_ATTRIBUTES,       pm_hdl_get_obj_attr   },
```

```
    {SYS_MSG_SET_OBJ_ATTRIBUTES,        pm_hdl_set_obj_attr     },
    {SYS_MSG_SET_COMMIT,                pm_hdl_set_commit       },
    {SYS_MSG_GET_SHM_INFO,              pm_hdl_get_shm_info     },
    {SYS_MSG_SERVICE_ACTION_RESULT,     pm_hdl_recv_act_res     },

};
```

配置管理模块对应的进程最终调用 mbus_msg_loop()接口进入消息循环，mbus_msg_loop 接口函数内部阻塞在接收 fd（文件描述符）上，当接收到消息数据时，当前调用此接口的配置管理进程解除阻塞并解析消息中的消息，与当前模块注册的消息进行逐一匹配，找到对应的处理函数并在当前进程的上下文中进行调用[27]。对应的所有消息处理函数在表 3-8 中列出：

表 3-8 配置管理模块消息处理函数列表

消息处理函数	说明
int pm_handle_set_param_val	处理：set value 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，并发送启动或重启服务的消息给服务管理模块，最后回复 response 消息给操作请求模块。
int pm_handle_get_param_val	处理：get value 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复确认消息给操作请求模块。
int pm_handle_get_param_name	处理:获取参数 name 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复 response 消息给操作请求模块。
int pm_handle_set_param_attr	处理：设置参数 attribute 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复 response 消息给操作请求模块。
int pm_handle_get_param_attr	处理：获取参数 attribute 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复 response 消息给操作请求模块。
int pm_handle_add_obj	处理：增加 object 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，并发送启动或重启 service 的消息给 service 管理模块，最后回复 response 消息给操作请求模块
int pm_handle_del_obj	处理：删除 object 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，并发送启动或重启 service 的消息给 service 管理模块。

表 3-8（续表）

消息处理函数	说明
int pm_handle_upload	处理：upload 的请求消息。最后回复 response 消息给操作请求模块。
int pm_handle_download	处理：download 的请求消息。最后回复 response 消息给操作请求模块。
int pm_handle_reboot	处理：系统 reboot 的请求消息。重启之后发送 inform 通知给 TR069 agent 模块。
int pm_handle_factory_reset	处理：恢复出厂设置的请求消息。最后回复 response 消息给操作请求模块。
int pm_hdl_get_obj_attr	处理：获取 object attribute 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复 response 消息给操作请求模块。
int pm_hdl_set_obj_attr	处理：设置 object attribute 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复 response 消息给操作请求模块。
int pm_hdl_set_commit	处理：设置 commit 的请求消息。内部需要调用各 feature 注册的回调函数，以及数据模型管理接口，最后回复 response 消息给操作请求模块。
int pm_hdl_get_shm_info	处理：获取 share memory 信息的请求消息。内部获取共享内存的地址、ID 等信息，通过回复 response 消息给操作请求模块。
int pm_hdl_recv_act_res	处理：service 管理模块的回复消息。包括：启动/重启/关闭 service 的响应结果，查询 service 状态的响应结果等。

3.5.2 配置处理成功流程的设计

如图 3-9 所示，描述了配置请求处理成功的时序。解析模块（Agent）接到外部 UI 的请求后，将该请求事件通过消息总线发送给配置管理模块，配置管理模块监听到该消息后调用相应的消息处理函数解析该消息。然后，根据解析的消息内容与动作更新临时配置文件，并且向服务管理模块发消息处理某些服务。服务管理模块处理相应的服务后（启动、停止、重启），相应的应用程序重新加载临时配置文件并向服务管理模块反馈运行的状态。服务管理模块接到该反馈结果后则发消息给配置管理模块告知相应的服务修改成功。当所有的服务均已成功启用，配置管理模块将该临时配置文件回写至 XML 格式的永久性存储配置文件中。最后，配置管理模块将处理结果反馈给解析模块，解析模块再

将该结果反馈给相对应的外部 UI。

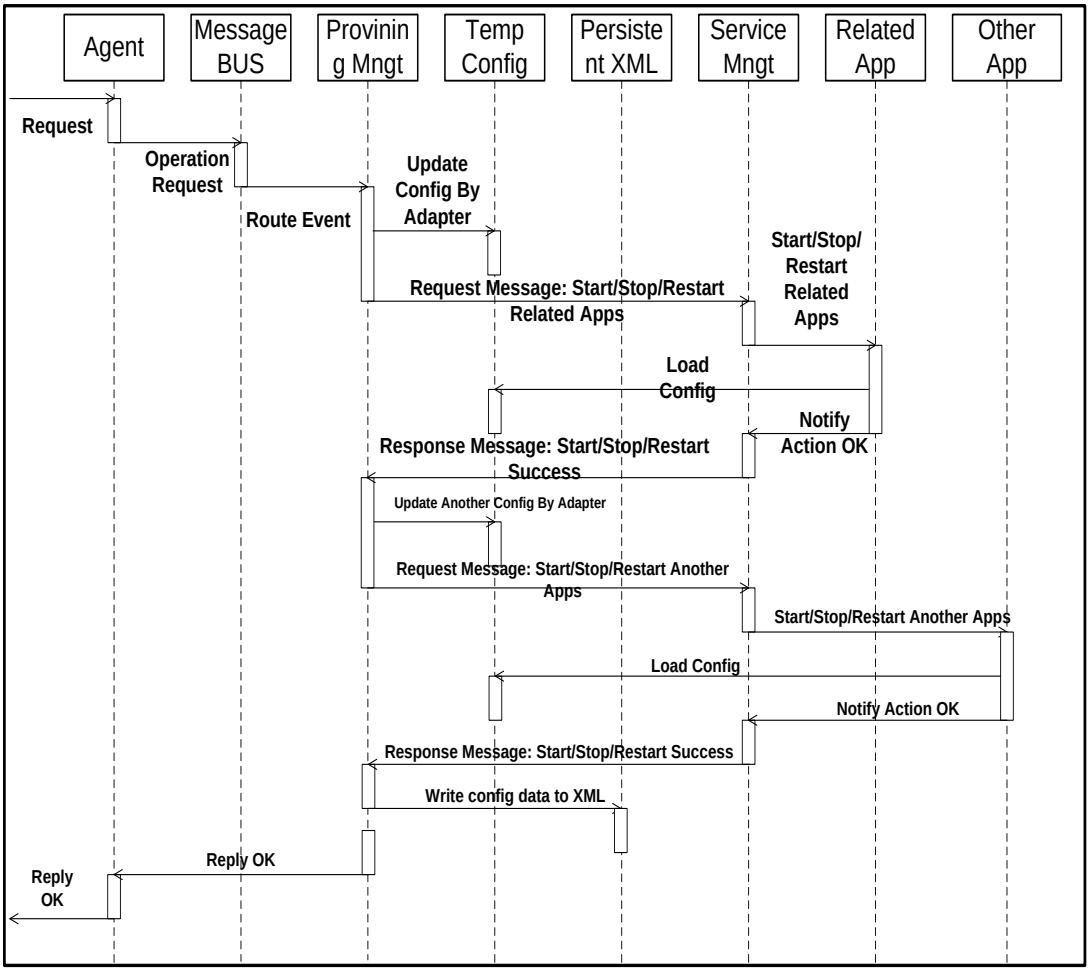

图 3-9 配置请求处理成功时序图

如果该修改配置的请求会修改多个服务，例如将 IP 地址设置成 192.168.1.6 并且关闭无线网络，IP 地址由 NETWORK 应用程序负责，无线网络由 WIRELESS 应用程序负责。那么，配置管理模块会依次更新临时配置文件并且服务管理模块也会依次重新加载各应用程序的服务。若所有的服务均已成功加载，配置管理模块才会将临时文件回写到永久性存储文件中。只要其中存在一个服务没有成功加载，则该修改配置的请求为失败。

3.5.3 配置处理失败流程的设计

图 3-10 描述了配置修改失败的时序。解析模块（Agent）接到外部 UI 的请求后，将该请求事件通过消息总线发送给配置管理模块，配置管理模块监听到该消息后调用相应的消息处理函数解析该消息。然后，根据解析的消息内容与

动作更新临时配置文件，并且向服务管理模块发消息处理某些服务。服务管理模块处理相应的服务后（启动、停止、重启），相应的应用程序重新加载临时配置文件并向服务管理模块反馈运行的状态。服务管理模块接到该反馈结果后则发消息给配置管理模块告知相应的服务修改失败。此时，临时配置文件中的配置信息已经受到污染，路由器已无法正常工作，配置管理模块将从永久性存储文件中恢复正确的配置到临时配置文件中。最后，配置管理模块发消息告知解析模块配置修改失败。解析模块将这一配置修改请求结果反馈给相应的外部UI。

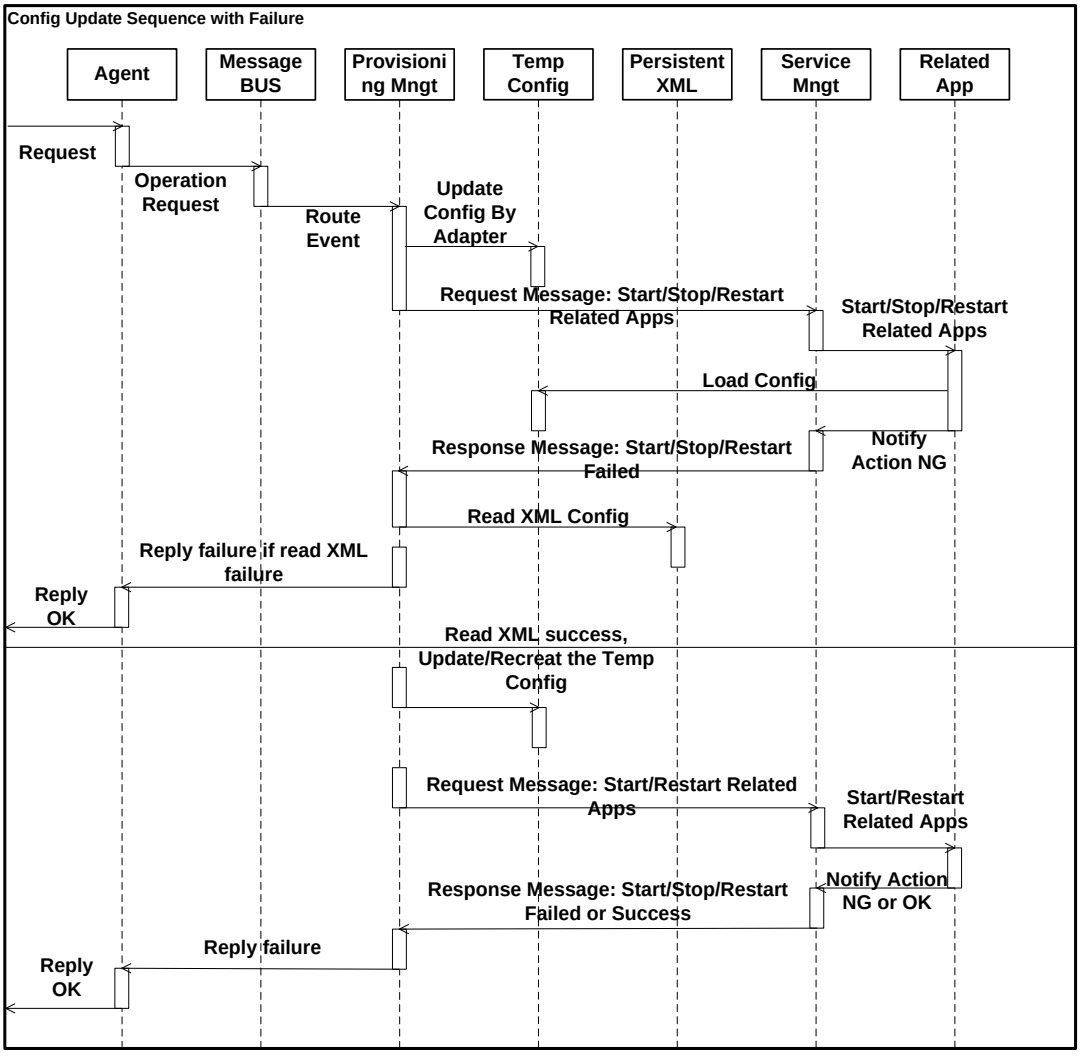

图 3-10 配置请求处理失败时序图

3.5.4 数据模型管理

数据模型提供了一个支持所有基于 TR-069 协议的 TR-098 数据模型的框架

[28]。定义的数据模型不仅仅用于配置管理模块，同时也用于任何其它服务程序访问本地临时配置文件的情况。在配置管理模块的启动过程中，将会加载永久的 XML 文件。在定义了这些数据模型的基础上，各外部接口向配置管理模块发消息读写内存，当需要同步到 XML 配置文件时，使用 XML 库的接口函数来实现[29]。该数据模型已经被定义完整，XML 库为配置管理模块提供了操纵数据模型的全部调用函数，配置管理模块只需调用数据模型库提供的接口函数即可实现对 XML 永久性存储文件的读写。数据模型已经为配置管理模块提供了可以直接使用的接口函数，配置管理模块直接调用数据模型提供的接口函数就可以实现对 XML 文件的直接操作[30]。

3.5.5 服务适配子模块的设计

蝴蝶路由器设备的永久性存储配置数据保存在 XML 文件中，相对应的各个服务所依赖的配置文件被称为临时配置文件。服务适配子模块的功能是完成对系统临时配置文件的操作。

配置管理模块在启动过程中，需要通过数据模型管理接口从 XML 配置文件中读取配置数据，并创建、更新各个服务所依赖的临时配置文件。或在运行过程中，用户通过各种 UI 解析模块（如 WebUI、TR069 ACS）读取、修改配置，则对应的 UI 解析模块发送配置访问请求消息给配置管理模块，然后由配置管理模块完成对临时配置文件的读取、修改操作。

建立适配层的目的，就是通过各服务向配置管理模块注册 Handler 函数的方式，对各个服务所依赖的临时配置文件进行创建、更新、读取。主要结构体定义如下：

```
typedef struct srv_config_adapter_t
{
struct srv_config_adapter_t *next;
char                          *file_name;
    SYS_APP_INFO                  *app_info;
short                         update_flg;
int                           param_cnt;
const char                    **param_list;
int     (*service_config_create)(struct srv_config_adapter_t *config_map)，
int     (*service_config_update)(struct srv_config_adapter_t *config_map,
                                 SYS_SERVICE_INFO **service_info,
                                 DM_CHANGE_REQ_T
```

*dm_chg_req)，

} SRV_CONFIG_ADAPTER_T;

结构体说明如表 3-9 所示：

表 3-9 SRV_CONFIG_ADAPTER_T结构体说明

类型	变量名	说明
struct srv_config_adapter_t	next	链表变量
char*	file_name	config 文件名
SYS_APP_INFO*	app_info	service 所含有的 app proc 信息
short	update_flg	0：update 失败 1：update 成功
int	param_cnt	参数列表中的参数个数。
constchar**	param_list	针对 service 的符合 tr098 数据模型的参数列表。
int (*service_config_create)(struct srv_config_adapter_t *config_map);		sevice 注册的配置文件创建 handler 函数。
int (*service_config_update) (struct srv_config_adapter_t *config_map, SYS_SERVICE_INFO **service_info, DM_CHANGE_REQ_T *dm_chg_req);		service 注册的配置文件更新 handler 函数。
} SRV_CONFIG_ADAPTER_T;		

3.6 本章小结

本章主要说明了对消息总线模块（M-Bus）、服务管理模块（Service Management）、配置管理模块（Provisioning Management）进行了详细设计过程。每个模块都给出总体设计，然后，设计了各模块的数据结构。

（1）设计了路由软件系统各个应用程序的先后启动顺序。

（2）设计了消息总线模块，总体设计该模块，然后给出全局消息的定义、消息组成的数据结构定义、对外接口等进行了的详细设计阐述

（3）设计了服务管理模块对系统中各进程的组织管理和维护各系统各服务的数据结构。

（4）设计了配置管理模块如何维护与管理本地配置，如何进行消息处理，以及对路由器软件系统的配置处理进行了设计。

第 4 章 路由器软件系统的实现

本章主要对课题消息总线模块、服务管理模块、配置管理模块实现过程的介绍，遵循前一章中系统中的详细设计按照顺序来说明。其中，消息总线模块主要介绍消息的注册、卸载、发送、接收、以及消息处理的实现过程；然后，服务管理模块主要从服务的注册、注销、启动、停止、重启以及子进程监管的具体实现来说明；最后，介绍了配置管理模块对配置操作的具体实现。

4.1 消息总线模块的实现

4.1.1 进程在消息总线上的注册与卸载

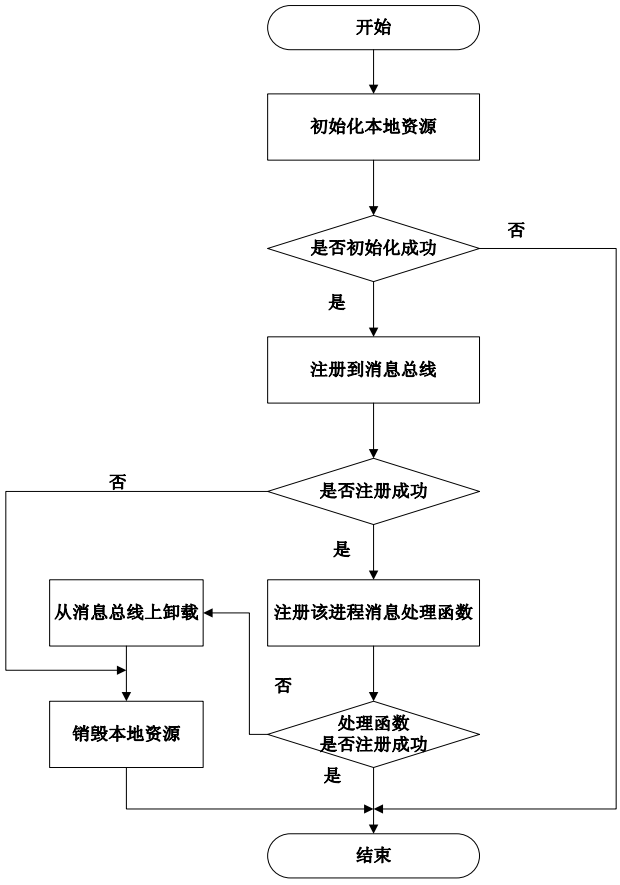

图 4-1 注册/卸载流程图

如图 4-1 所示：路由器内某各应用程序若想与其他应用程序实现数据的传

递（进程间通信），首先需要初始化本地相关资源，向内存申请一块空间来存放消息上下文，建立 M-Bus 使用日志目录的操作。然后，将自身进程名字注册到消息总线上。当该进程不需要使用消息总线的时候，调用 M-Bus 卸载 API 函数从 M-Bus 上卸载并且释放消息上下文使用的内存空间。

该流程图描述了一个进程注册到消息总线上的全过程：

（1）首先调用 mbus_init 函数初始化本地资源并生成日志目录，如果初始化成功则调用 mbus_obj_register 函数注册到消息总线，否则返回失败，然后结束并退出。

（2）调用注册函数会生成该进程的消息上下文，指明进程从消息总线上的退出函数，该进程会以一个对象的身份（Object）注册到消息总线上。如果注册失败，则调用 mbus_terminate 函数销毁本地资源，然后结束并退出；如果注册成功，则调用 mbus_register_event_map 函数登记该进程的消息处理函数。消息处理函数是指一个进程收到不同的消息后所执行的相应处理程序[31-32]。消息处理函数与其对应的消息用一个数据结构定义，该数据结构定义如图 4-2 所示：

MBUS_EVENT_MAP

处理的消息类型 int Event_type	消息处理函数 mbus_event_handler event_handler

图 4-2　处理函数指针数据结构示意图

Event_type 数据域表示消息的类型，mbus_event_handler 数据类型是被重新定义的新类型，event_handler 被定义为消息处理函数的指针。使用消息总线的进程可以使用 MBUS_EVENT_MAP 数组，指明消息类型与消息处理函数的对应关系，一次注册多个消息处理函数。

（3）如果消息处理函数注册成功，则返回 0（表示注册成功）；如果消息处理函数注册失败，则说明无法构建消息与消息处理的关系。那么，注册进程就会调用消息总线卸载 API 函数 mbus_obj_unregister，将注册进程从消息总线上卸载。然后调用 mbus_terminate 函数销毁本地资源，释放内存空间。最后退出并结束。

4.1.2 发送消息的实现

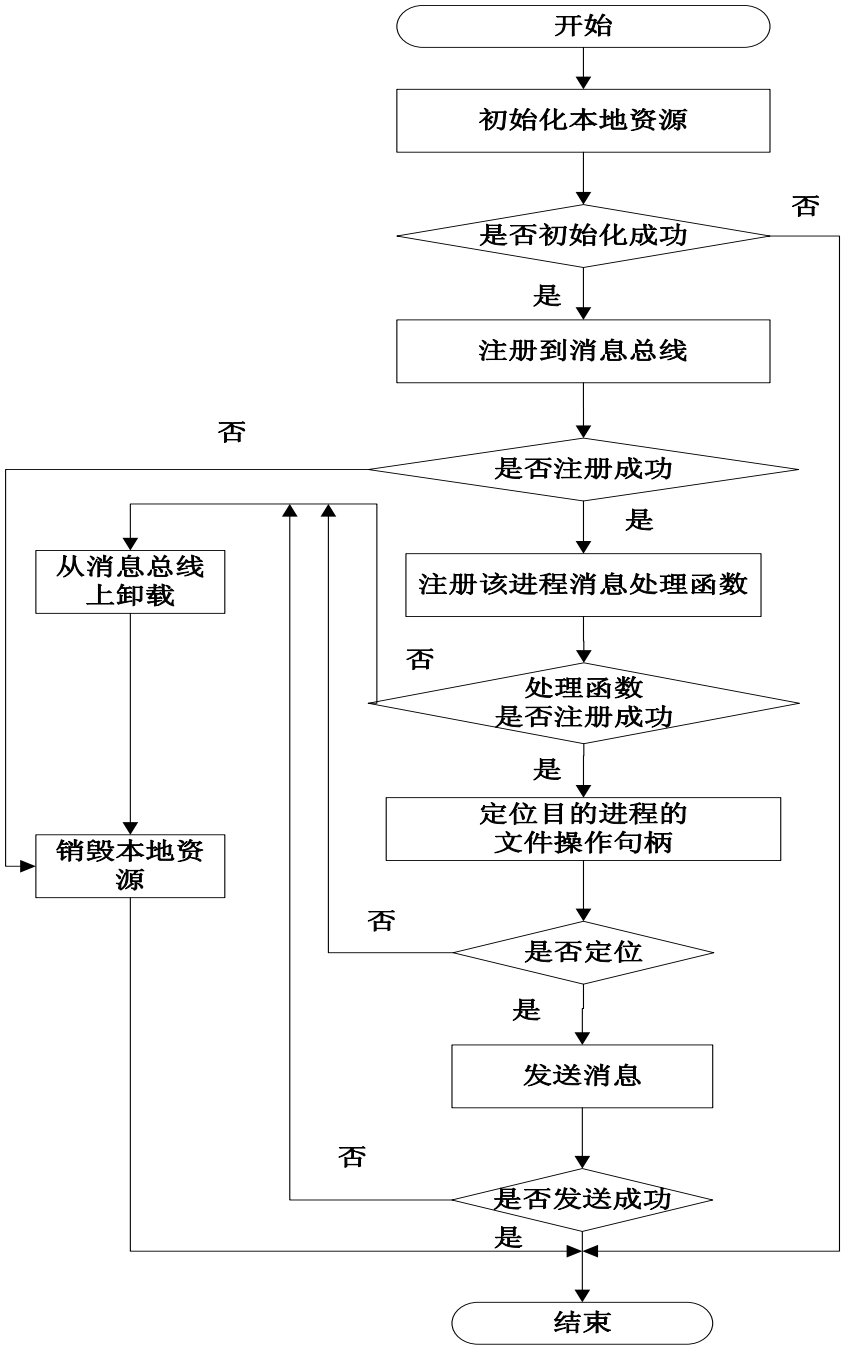

图 4-3 消息发送流程图

图 4-3 描述了一个进程从注册到发送消息的全过程。注册进程首先初始化本地资源，然后注册到消息总线，接着注册消息处理函数。如果消息处理函数注册成功，则定位目的进程的文件操作句柄，目的进程是消息的接收进程[33]。最后，调用 mbus_send 函数发送消息。如果消息发送成功，则返回发送的数据

大小；如果消息发送失败，该进程将从消息总线上卸载，释放本地资源，最后结束。

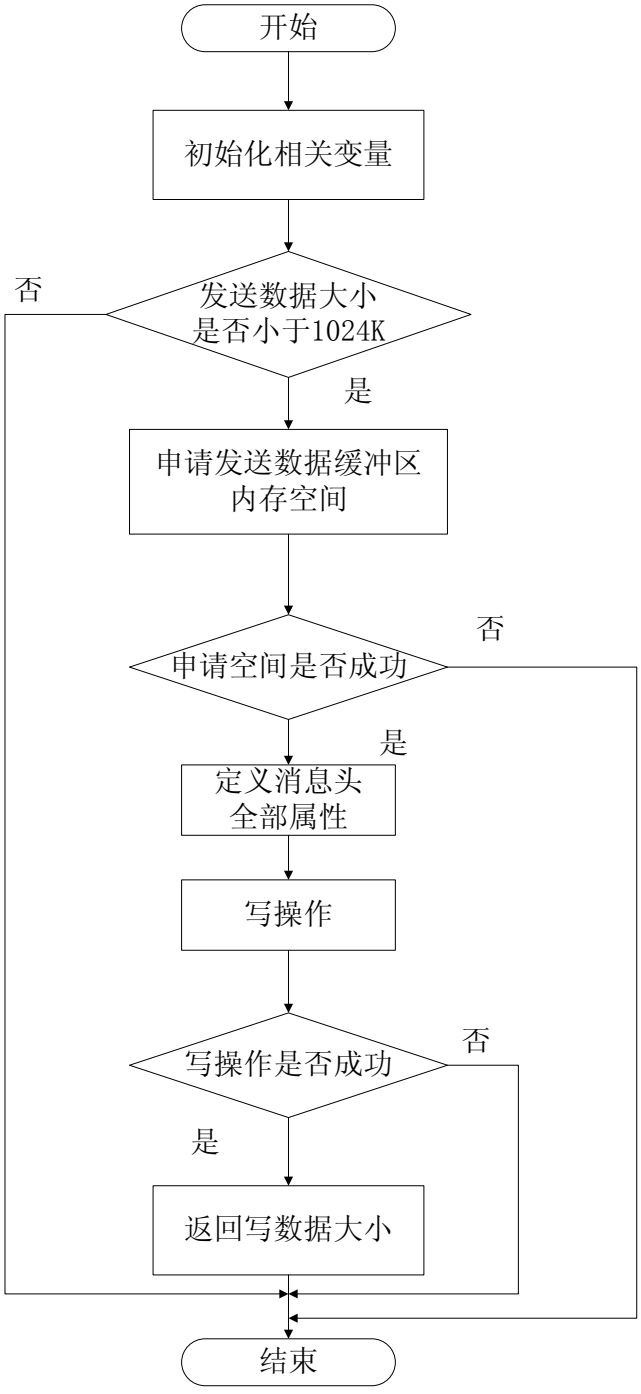

图 4-4　发送消息函数流程设计

图 4-4 是图 4-3 中的发送消息节点的具体实现，是 mbus_send API 函数的设

计过程。发送消息函数的具体过程如下：

（1）初始化相关变量：定义消息头指针、消息上下文指针、发送数据缓冲区、创建命名管道、指明消息接收进程的文件操作句柄。

（2）判断所发数据大小是否在允许范围之内。数据最大为宏定义MBUS_MAX_MSG_LEN 个字节，该宏定义对应的最大发送字节数为 1023 字节（十六进制为 0x3FF）。如果所发字节数量小于或等于 MBUS_MAX_MSG_LEN，则定义消息头。否则，函数返回失败并结束。

（3）消息头中将定义如下属性： 消息的发送者、消息长度、是否同步、消息类型、消息携带数据的起始地址。

（4）消息头定义完毕后，将消息头的内容与携带数据使用内存拷贝函数复制到发送缓冲区中。然后进行写操作，将发送缓冲区的内容写到消息接收进程的文件描述符下。

（5）如果写操作成功，函数将返回发送数据的大小并结束，否则，返回失败（函数将返回-1）。

4.1.3 接收消息的实现

图 4-5 描述了消息接收 API 函数 mbus_recive 的具体实现过程：

（1）打开命名管道。

（2）从命名管道上等待消息。如果收到消息，则解析收到的消息头，并从消息头中获取该消息头中标识的数据长度。如果没有收到消息，函数返回失败并结束。

（3）解析消息头的携带数据起始地址，并从起始地址指针开始读取数据（读操作）。

（4）判断所读数据的大小是否与消息头中标识的数据长度相等，如果相等，则说明所读取的数据完整。如果所读数据的大小与消息头中标识的数据长度不相等，则说明所读的数据有丢失，读取的数据不完成，函数返回失败并结束。

（5）判断所得到的该消息是否需要同步，同步是指接收进程收到消息后需要将成功收到消息的这一事件告知发送进程。该消息是否需要同步，在消息头中的同步标志位中表示。0 为不需要同步，1 为需要同步。如果该消息需要做同步操作，则从消息头中记录发送进程的相关信息，给发送进程回消息，然后关闭命名管道。如果不需要同步，则直接关闭命名管道。

（6）返回接收到的消息并结束。

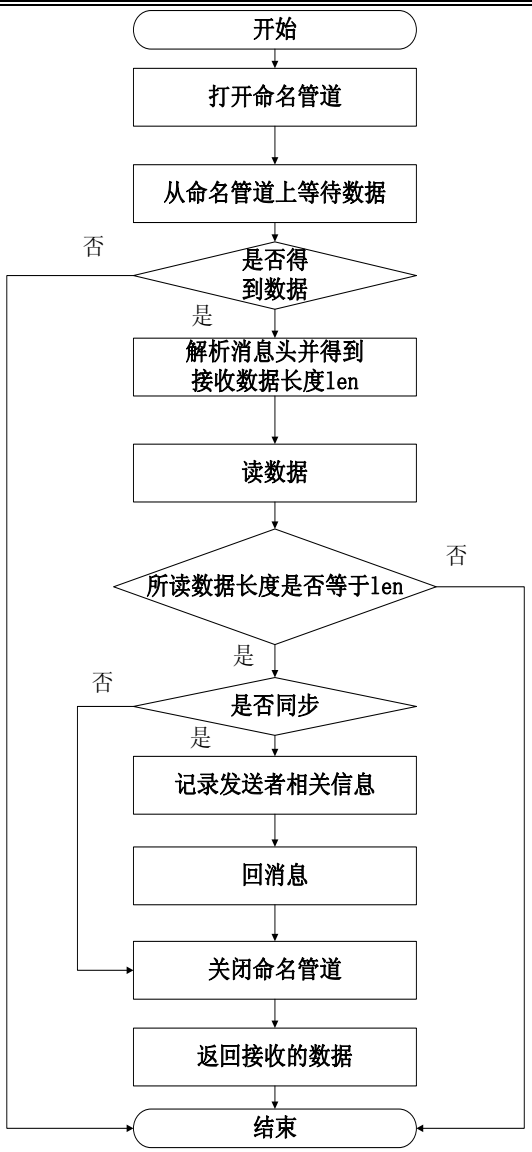

图 4-5　接收消息函数流程设计

图 4-6 描述了一个进程从注册到接收消息的总体过程（接收一次消息的过程）。首先，该进程初始化本地资源并注册到消息总线上，并且将该进程相应的消息处理函数进行注册。然后，程序进入无限循环，调用 mbus_receive API 函数进行消息的接收。如果收到的数据大于 0，则解析该数据，提取其中的消息类型并调用与该消息对应的消息处理函数进行处理。如果收到的数据小于或等于 0，进程将休眠 100 微秒，然后再次接收消息。

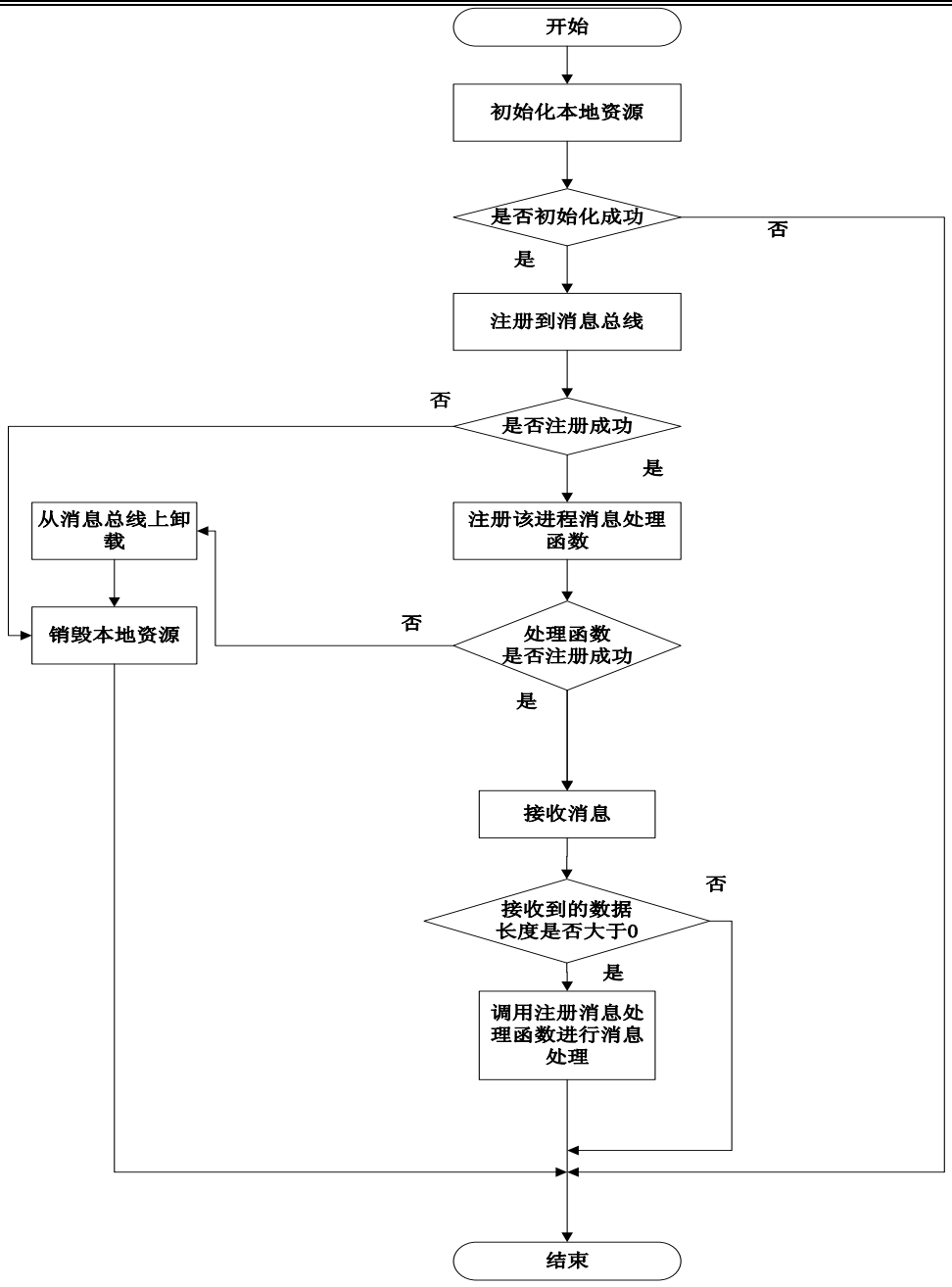

图 4-6　消息接收流程图

4.2 服务管理模块的实现

4.2.1 注册服务的实现

图 4-7 描述了服务注册的整个工作流程。服务管理模块的主进程启动之后首先注册到消息总线上,然后循环监听配置管理模块通过消息总线发来的消息。

当消息总线上注册服务 API 函数被调用后，会向服务管理模块发送服务注册请求消息。服务管理模块收到请求后就会解析消息，注册服务的详细流程如下：

（1）解析收到的数据，从中解析服务名。

（2）查询本地服务列表。

（3）判断该服务是否被注册。

（4）如果该服务已经被注册则返回 FAILED 结果给配置管理模块。如果该服务没有注册过，则在服务列表中增加该服务的节点。

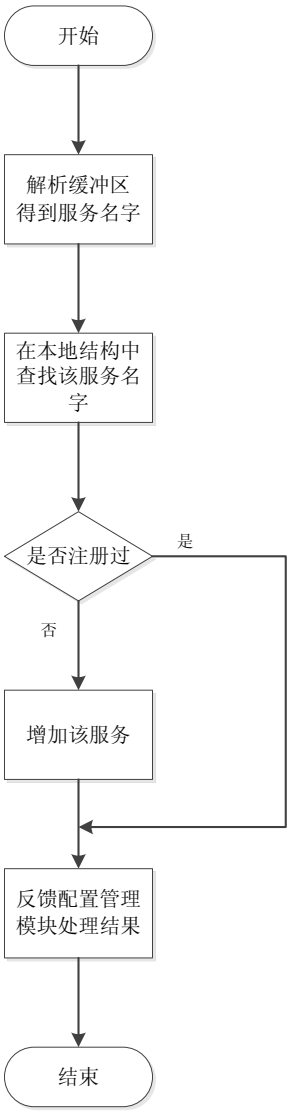

图 4-7 服务注册流程图

4.2.2 卸载（注销）服务的实现

图 4-8 描述了卸载某个服务的流程：

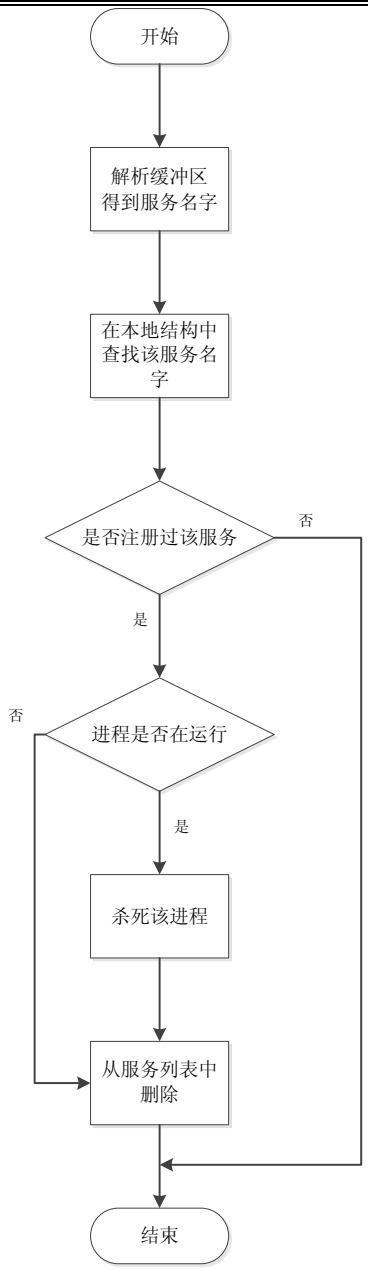

图 4-8 服务卸载（注销）流程图

配置管理模块调用服务管理模块提供的卸载 API 函数，通过消息总线将消息发送给服务管理模块。服务管理模块收到消息后则解析该消息，卸载服务的详细流程如下：

（1）若解析的消息为“卸载某服务”，则解析所卸载服务的名字。

（2）查询本地服务列表是否存在该服务的名字。

（3）如果本地服务列表中没有该服务的名字，则说明本地并为注册过这个服务因此无法卸载该服务，然后直接返回卸载失败的结果。

（4）如果本地服务列表中存在该服务的名字，则判断该服务所对应的应用程序进程是否在运行。

（5）如果该服务对应的应用程序进程没有运行，则从本地服务列表中删除该服务的节点。

（6）如果该服务对应的应用程序进程还在运行，则先调用系统函数杀掉该服务对应的进程。然后再从本地服务列表中删除该服务的节点。

（7）判断是否删除服务节点成功，如果删除节点成功则返回给配置管理模块卸载服务成功的结果。否则返回卸载失败的结果。

4.2.3 启动服务的实现

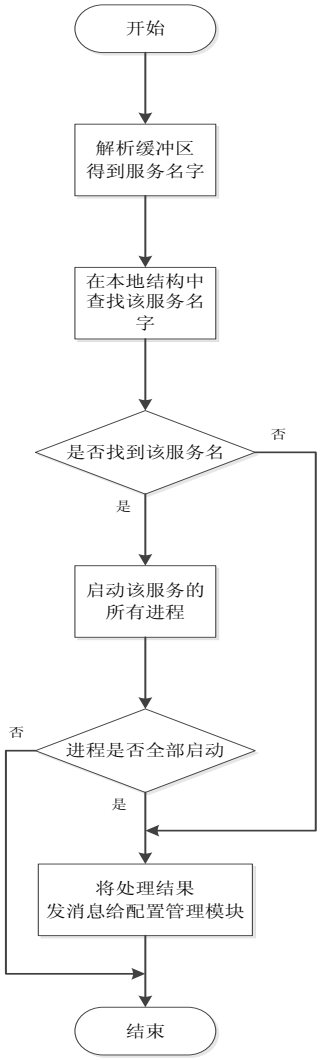

图 4-9 启动服务流程图

图 4-9 描述了服务卸载的详细流程。

配置管理模块调用服务管理模块所提供的启动服务 API 函数，当服务管理模块收到该消息时会解析该消息，启动服务的流程如下：

（1）解析该消息并从中提取服务的名字。

（2）查询本地服务列表是否存在该服务的名字。

（3）如果没有找到该服务的名字，则返回启动失败的结果发送给配置管理模块。

（4）若果本地服务列表中存在该服务的名字，则由服务管理模块通过系统调用启动该服务对应的应用程序进程。

（5）判断该服务对应的所以进程是否全部启动成功，如果成功则向配置管理模块返回服务启动成功的结果，否则返回失败。

4.2.4 重启服务的实现

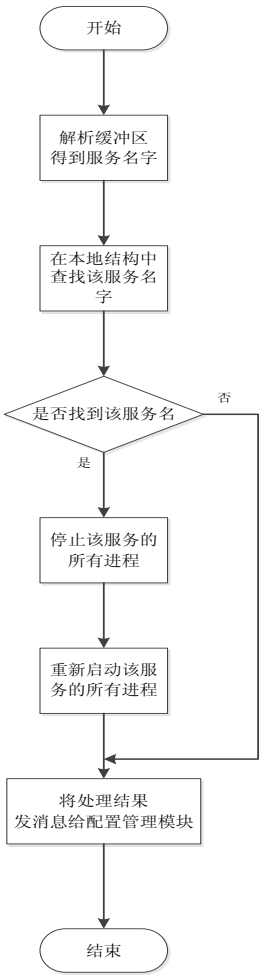

图 4-10 重启服务流程图

图 4-10 描述了重启服务的详细设计流程。

配置管理模块调用重启 API 函数后，会向服务管理模块发送服务重启请求消息，具体流程如下：

（1）解析消息并提取消息中要重启的服务名字。

（2）查询本地服务列表中是否存在该服务名字。

（3）如果没有找到该服务的名字，向配置管理模块返回重启失败的结果。

（4）如果本地服务列表中存在该服务的名字，服务管理模块就会通过系统调用停止该服务对应的所有进程。然后再通过系统调用启动该服务对应的所有进程。最后返回给配置管理模块服务重启成功的结果。

4.2.5 停止服务的实现

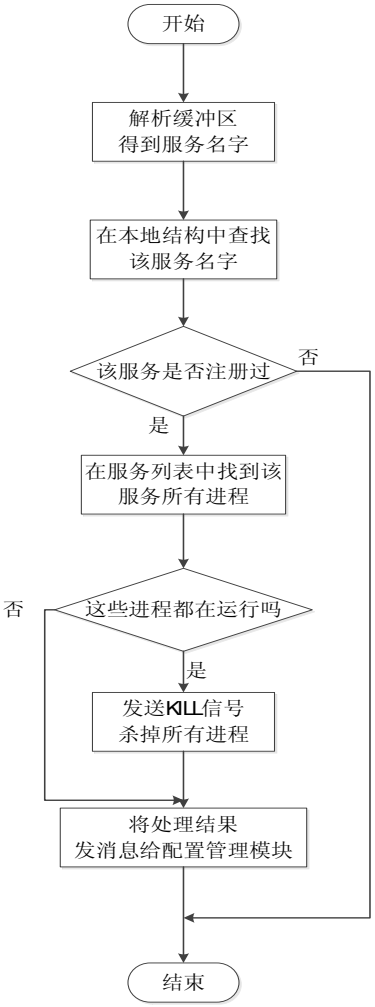

图 4-11 停止服务流程图

图 4-11 描述了服务停止的详细流程。

配置管理模块调用停止服务 API 函数后，会向服务管理模块发送停止某服务的请求消息。具体流程如下：

（1）解析消息并提取消息中要停止的服务名字。

（2）查询本地服务列表中是否存在该服务名字。

（3）如果该服务并不存在（该服务从来没有注册过），则返回服务停止失败的结果给配置管理模块。

（4）如果该服务存在，则找到该服务对应的所有应用程序进程[34-35]。判断该服务对应进程是否在运行，如果没有运行则返回“停止成功”的结果给配置管理模块。否则，服务管理模块通过系统调用杀掉该服务对应的所有进程。

（5）如果该服务对应的进程还在运行，则返回“停止失败”的结果给配置管理模块，否则返回成功。

4.2.6 子进程管理的实现

图 4-12 与图 4-13 描述了服务管理模块对各服务下子进程监控的详细流程。当应用程序的主进程启动子进程时，向服务管理模块发送通知消息来注册。服务管理模块内部通过一定的数据结构（数据结构的定义参照 2.4.3 章节）管理各个应用程序主进程及其子进程。另外服务管理模块启动定时处理，读取 Linux 系统中/proc 目录下的信息，不断更新本地服务下的各进程信息[36-37]。子进程的管理分为两个方面，一方面是增加一个子进程，另一个方面是减少一个子进程。

1）增加一个子进程管理的详细设计流程入下：

（1）当某个应用程序的主进程（父进程）创建子进程时，子进程将自身 PID（进程号）和 PPID（父进程的进程号）告知服务管理模块。

（2）服务管理模块收到消息后解析该消息，然后遍历本地服务列表，并且继续查找该服务节点下对应的进程数组，然后再数组中找到与该子进程对应的 PPID 相同的进程节点。

（3）服务管理模块判断该子进程是否运行，如果该子进程正在运行则在与其对应的子进程管理链表中增加一个子进程节点（数据结构的定义参照 2.4.3 节）。

2）删除一个子进程的详细设计流程如下：

（1）当某个应用程序的子进程退出时，子进程将自身 PID（进程号）和 PPID（父进程的进程号）告知服务管理模块。

（2）服务管理模块收到消息后解析该消息，然后遍历本地服务列表，并且继续查找该服务节点下对应的进程数组，然后再数组中找到与该子进程对应的PPID 相同的进程节点。

（3）服务管理模块从本地服务列表中删除该子进程的节点信息。

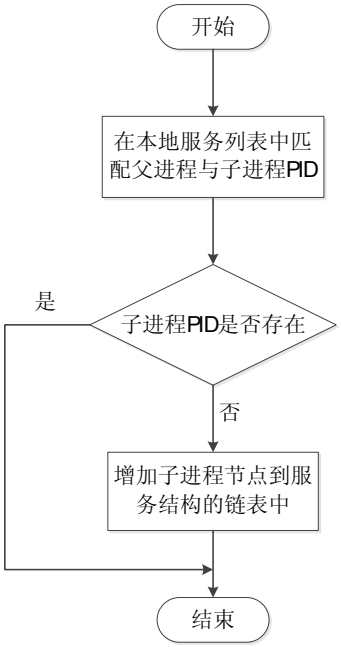

图 4-12 增加子进程节点流程图

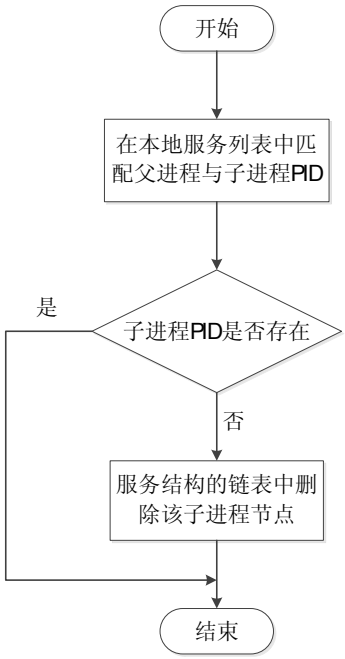

图 4-13 删除子进程节点流程图

另外，一方面：服务管理模块对于孙子进程（子进程创建的进程）管理采

用定时遍历的方法，服务管理模块定时读取系统中/proc 目录下的信息，与本地服务列表进行匹配，如果存在更新，则更新本地服务列表。另一方面：当一个进程在运行时出错突然出错退出，在这种情况下无法通知服务管理模块，依然使用定时遍历的方法实现对各进程的管理。

4.3 配置管理模块的实现

4.3.1 加载本地永久性配置的实现

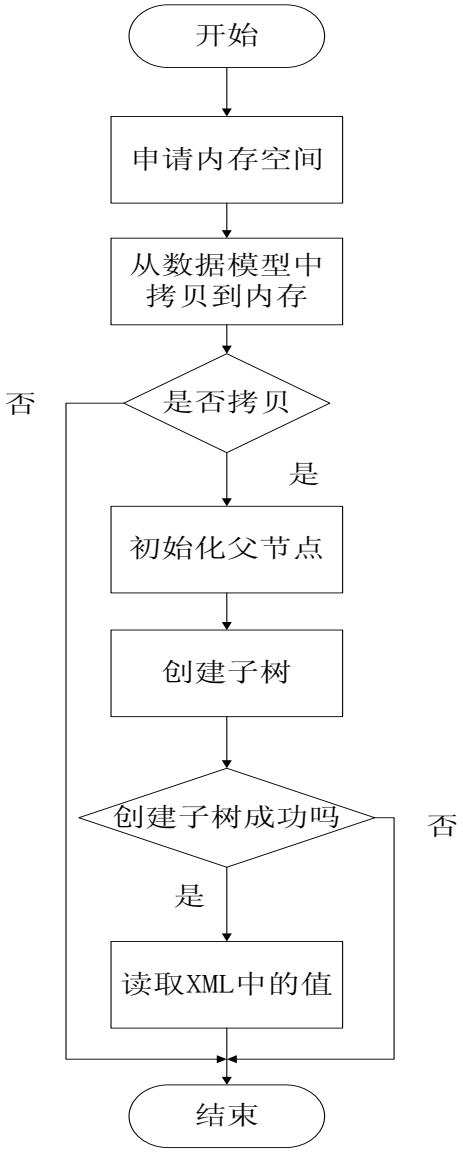

图 4-14 加载本地永久性配置流程图

路由器系统中的永久性存储配置保存在 XML 格式的文件中，由数据模型

统一管理与维护，路由器系统开机启动后，需将 XML 文件中的配置读入内存中，XML 文件存储路由器配置的结构是一个树形结构，每棵子树对应着存放一组路由功能，而每棵子树的叶子节点存放着该组路由功能的对应参数与数值。在 XML 文件中，将 object 定义为一组路由功能，路由功能下又分为 value 和 parameter。object 是子树的根节点，其对应的叶子节点分别为 value 和 parameter。

流程图 4-14 描述了加载本地永久性配置的实现过程。首先，向系统中申请内存空间，然后将数据模型中的数据拷贝到内存空间中。如果拷贝成功，初始化父节点并创建子树，若子树创建成功，则将事先读入到内存中的 XML 值写入到子树当中，最后函数结束返回。成功返回 0，失败返回-1。

4.3.2 系统配置写操作的实现

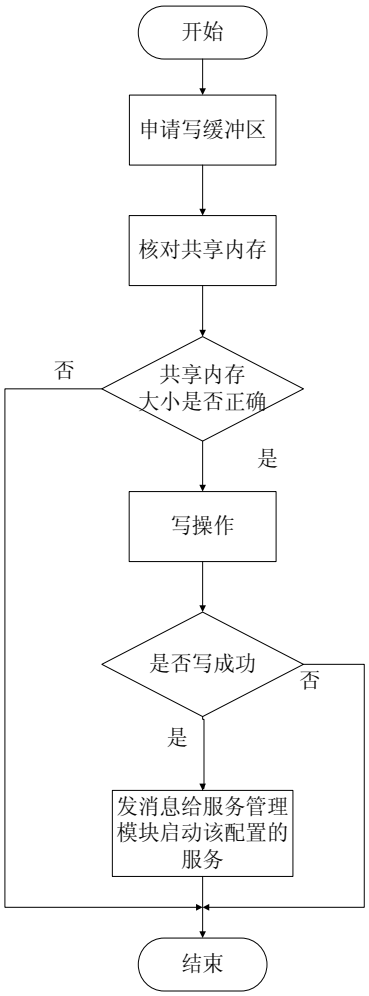

图 4-15 系统配置写操作流程图

用户对路由器配置进行修改的操作，会改变路由器的功能。修改配置的具

体实现使用了共享内存的方法。首先，申请缓冲区用于存放要写的配置参数，格式是字符串。然后创建共享内存检查该区域内存大小是否正确。若正确则进行写操作，并判断是否写入成功。最后发消息给服务管理模块重新启动修改配置后的各项服务。服务管理模块会调用相应的消息处理函数重新启动各项服务。图 4-15 描述了这一环节的实现过程。

4.3.3 系统配置读操作的实现

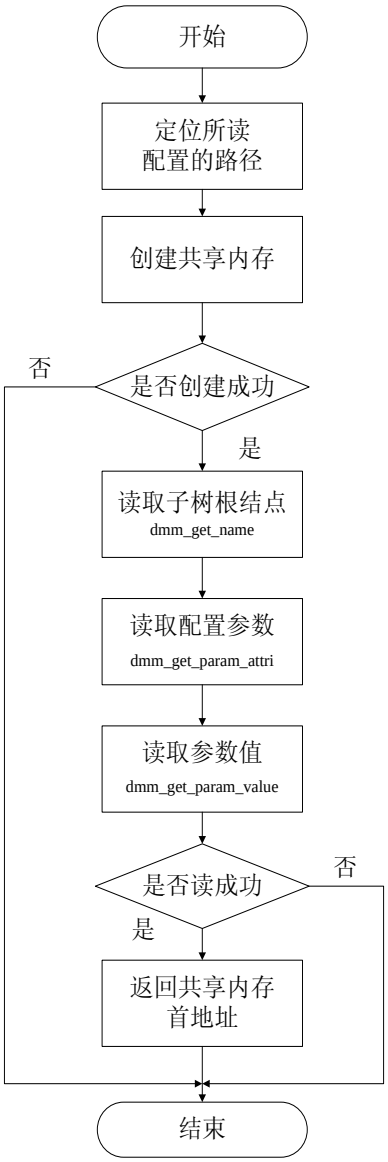

图 4-16 系统配置读操作流程图

配置管理模块是从数据模型中读数据，从而实现系统配置的读操作。首先，

定位所读配置的路径，然后在内存空间中开辟出一块共享内存，进行读配置名字的操作，例如，LAN、WAN、WLAN 等统一并独立的功能点均属于配置名字，它们在数据模型中以一个 Object 的形式存在（一棵子树的根）。最后将该名字对应的数值和参数（子树的叶子）也进行读取，从而完成整个读操作。数据管理模型提供了已有的两个读操作接口，分别是读配置名字的接口函数（dmm_get_name）和读配置参数（dmm_get_param_attri）与数值（dmm_get_param_value）的接口函数。流程图 4-16 描述了配置管理模块读操作的实现过程。

4.4 本章小结

本章主要按照第三章的设计过程介绍了消息总线模块、服务管理模块、配置管理模块的实现过程。主要展示了程序中关键函数处理的流程，通过阐述关键函数的处理流程，来反映系统各模块的主要功能是如何实现的。

第 5 章 路由器软件系统测试与评价

软件测试是软件开发流程中的关键步骤，通过软件测试从而验证模块和系统中各项功能的完成性与正确性，以及产品性能满足客户的需求要求[38-39]。本章主要说明蝴蝶路由器软件系统的测试过程，通过测试的结果，对设计与实现的工作进行有效的评估。

5.1 测试方法和测试环境

在路由软件系统设计和实现的过程中，已经采用了白盒测试的方法对各个功能模块、核心代码进行了验证与详细走查，可以保证测试用例覆盖的完整性，以及核心代码的正确性。后期开发环节里，主要采取黑盒测试的方法，对路由软件系统各功能模块与整机进行覆盖测试。在本课题中，虽然路由器界面不在课题研究范围之内，但是，在黑盒测试中主要通过路由器的 WEB 界面来测试路由器的各项功能，从而验证消息总线模块、服务管理模块以及配置管理模块是否正常工作。且能够正确的跳转，符合用户提出的全部需求。并且路由器软件系统的各项功能均满足用户的要求。

嵌入式系统的测试需要硬件平台的支撑，因此本课题先在计算机上运行整个路由器软件系统，最后，再进行整机测试，测试系统整机的功能和性能。

模拟测试环境：

（1）OpenWrt 平台，Linux 内核版本 2.6.38。

（2）具备 WiFi 功能的手持设备一部。

（3）Intel EIG44HT PCI-E/I340-T4 四口以太网卡。

（4）Intel(R) PRO/1000 MT 82546EB 双口以太网卡。

（5）通用支持 802.11b/g/n 混杂模式 WiFi 网卡。

（6）HUAWEI Surfing 3G 上网卡。

（7）各种硬件硬件设备的接口线。

整机测试环境：

嵌入式平台选择 Hewlett-Packard Vo1.01 芯片功能套件。

5.2 功能测试

5.2.1 WEB 配置界面功能测试

通过对 WEB 界面的功能测试，验证路由器界面显示与跳转流程的正确性，测试的范围涵盖路由软件系统所有配置界面。列举若干典型的测试用例和相关界面如下，VLAN 管理功能测试用例如表 5-1 所示。

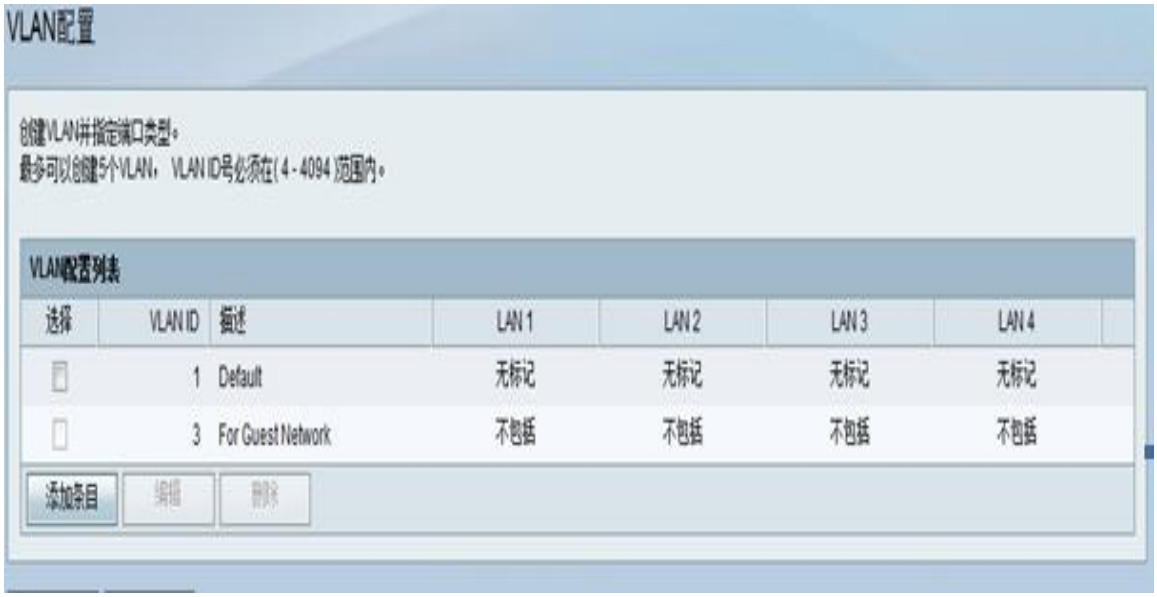

图 5-1　VLAN 配置界面

表 5-1 VLAN 功能测试用例

用例描述	进入路由器 WEB 配置界面中的 VLAN 设置
测试类型	模块功能测试
基本条件	路由器系统启动，输入 WEB 网址进入 WEB 管理页面
测试流程	1、进入网络配置界面 2、进入 VLAN 接口配置界面 3、浏览 VLAN 基本配置列表 4、添加 VLAN 条目信息 5、编辑 VLAN 条目信息 6、删除 VLAN 条目信
预测结论	在 VLAN 配置中，可以实现对 VLAN 的配置。

如图 5-1 所示，测试结果与预测结论相同，在 VLAN 配置中可以实现相应的配置处理，保存配置后生效。

DHCP 功能测试用例如表 5-2 所示。

表 5-2　DHCP 功能测试用例

用例描述	DHCP 功能的相关操作
测试类型	功能测试
先前条件	路由器系统启动，输入 WEB 网址进入 WEB 管理页面
测试流程	1、进入 DHCP 管理界面 2、开启 DHCP 功能 3、DHCP 最大用户数设置值为 50 4、DHCP 起始分配 IP 地址设置为 192.168.1.100
预测结论	各个界面成功跳转，连接到 LAN 口的计算机可自动分配到 IP 地址

测试结果与预期结果基本一致，各界面跳转功能正常，客户机可以分配到正常的 IP 地址。界面展示如图 5-2 所示：

图 5-2　DHCP 功能界面展示

路由器无线设备测试用例如表 5-3 所示。

表 5-3　无线功能测试用例

用例描述	测试无线功能
测试类型	功能测试
基本条件	输入网址进入 WEB 管理页面
测试流程	1、进入无线配置界面 2、启用无线 3、设置信号强度 4、设置工作模式 5、设置工作频段 6、信道选择
预测结论	可获取 SSID，无线设备连通后可以分到 IP 地址。

通过实际测试验证，与预测结论一致。信号强度、工作模式、工作频段、信号选择等功能均可实现并正常工作，手持设备可搜索到路由器的 SSID，并可连接分到 IP 地址。界面展示如图 5-3 所示：

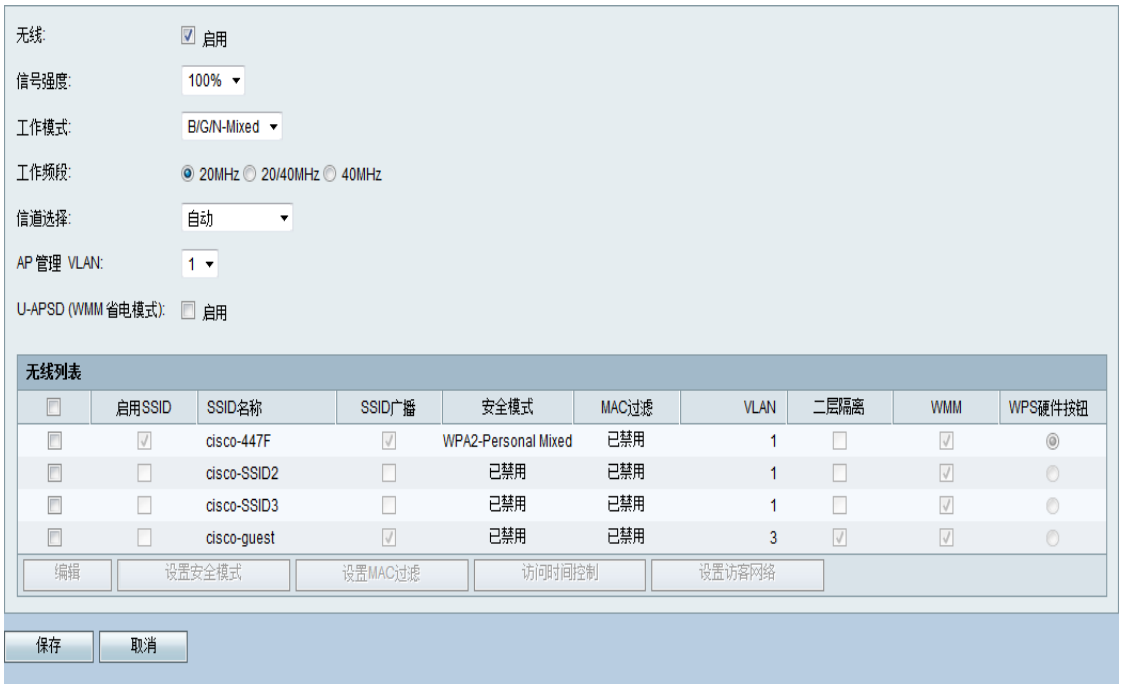

图 5-3　WIFI 功能界面展示

路由器 WAN 功能测试用例如表 5-4 所示。在 WAN 功能操作界面中进行相应的配置修改操作，用户可以通过 PPPOE、DHCP 自动配置、手动配置 IP 的方式连接外网。通过测试所得的结果与预期结论完全相同，WAN 功能界面展示如图 5-4 所示。

图 5-4　WAN 功能界面展示

表 5-4 WAN 功能测试用例

用例描述	测试 WAN 功能的相关操作
测试类型	功能测试
基本条件	路由器系统启动，输入网址进入 WEB 管理页面
测试流程	1、进入 WAN 配置界面 2、设置 WAN 联网方式
预测结论	各个界面成功跳转，系统参数可正确设置并显示，路由器可连接外网。

5.3 性能测试

为了满足客户所提出的硬性需求，还要进行整机性能测试。性能测试是对路由器软件系统在实际硬件平台上运行时进行测试。性能测试是指软件系统移植到整机之后，对产品的一致性、稳定性、工作功耗、开机时间等方面的测试[40-41]。其中，需求中要求系统运行时最大内存（RAM）使用量小于 32MB；系统 ROM 小于 8MB；新增消息总线、服务管理模块、配置管理模块总存储小于 650KB。测试的方法是将路由器软件系统移植到惠普的 Hewlett-Packard Vo1.01 芯片功能套件上。表 5-6 是经过性能测试后得到的测试结果。

表 5-6 路由器软件系统性能测试

测试点	测试方法	测试结果
网络性能测试	使用 PING 命令	Reply from 10.48.128.1: bytes=32 time=1ms TTL=50 性能无下降
稳定性测试	连续运行 168 小时	运行正常，反应速度没有下降
开机启动时间测试	接通电源正常启动 20 次	系统平均启动时间 60 秒以内
WAN 测试	PPPOE 拨号 100 次	接续成功率 99%
DHCP 测试	客户机 DHCP 分配 IP 地址 100 次	可正确获得 IP 地址
功耗测试	测量路由器时的输出电压和电流，计算功率	正常工作功耗约 3.2 瓦
系统最大内存使用量	开启路由器所有功能并使路由器负荷工作	内存使用 30.5MB
ROM 使用情况	将代码移植到惠普平台并核算实际大小	ROM 使用 7.8MB

5.4 测试结果分析与评价

通过以上的测试结果可以得到以下结论：本课题路由器软件系统功能点齐全，各功能界面跳转正确，系统开机与反应速度在客户的允许范围以内，满足客户需求。从而证明了新增的消息总线模块、服务管理模块、配置管理模块正常工作，代码运行良好。最后，进行了整机的性能测试。该路由器系统的一致性和稳定性良好，系统平均启动时间在 46.5 秒，各项指标均很好的满足客户的规定和要求，消息总线模块可以完成各应用程序的通信，服务管理模块能够正常完成对各服务的管理，配置管理模块实现了对路由器系统中配置管理操作。路由器软件系统整体的优化设计工作基本合格。目前，该路由器的功能还在继续完善中。

5.5 本章小结

本章重点使用黑盒测试的方法，对路由器软件系统各功能模块进行了功能测试，以及对整机进行了性能测试。其中，功能测试包括了对路由器软件系统 WEB 界面中的各功能点进行了测试，配置生效验证了本课题新增的消息总线模块、服务管理模块、配置管理模块运行良好，配置修改操作可以实现。另外，通过整机测试的各项结果均符合产品的要求。

结　论

本论文通过对国内外 OpenWrt 平台应用、改造、优化方面的研究，并深入调查了市场上常见的路由器产品，阅读了大量 OpenWrt 软件系统的文档，优化设计并实现了消息总线模块、服务管理模块、配置管理模块，从而改造并优化了原有 OpenWrt 路由器软件系统。取得的主要成果有以下方面：

（1）从软件工程的角度上，运用了模块化的方法设计了路由器软件系统的整体架构，在 OpenWrt 平台上设计并实现了蝴蝶路由器软件系统。模块化的思想保证了消息总线模块、服务管理模块、配置管理模块以及其他路由器功能模块层次清晰，数据流向可控、可观；详细完成了产品的需求分析，并将产品各功能点覆盖到各个子模块，良好的完成了对该研究课题的模块划分。

（2）深入研究了 Linux 进程间通信的各种方法，采用命名管道的方式实现了进程间通信，实现路由器系统中的各模块信息传递。并定义了全局消息，各模块可通过设计的消息总线实现各模块在消息总线上的注册、卸载、消息发送以及消息的接收。这也做到了系统各模块的低耦合。

（3）在 OpenWrt 平台下设计并实现了服务管理模块与配置管理模块，弥补了原有 OpenWrt 平台没有对配置操作统一管理和对系统中各服务的统一管理这些缺陷。目前，可实现对系统中各服务的启动、停止、监控，以及对配置文件操作的统一管理功能。

无线路由器已经成为人们生活中的必需品，本课题设计并实现了基于 OpenWrt 平台的路由器软件系统，该路由器软件系统仍然具备很大的功能扩展能力。总结本课题，今后的研究方向有以下几个方面：

（1）对关键代码进行整体优化，对系统资源开销较大的程序进行改善，设计更好的消息组成的数据结构以降低内存开销。

（2)结合当今路由器市场形式,对路由器软件系统的需求进行更好的理解，增加现有路由器软件系统的功能，使其功能得到扩展而得以提高产品在市场上的竞争力。

（3）设计更好的人机交互界面，使得系统更加人性化、智能化。

参考文献

[1] Bird Intern Articles on Routing Software Openwrt[M].Hephaestus Books，2011：115-120.

[2] Jim Brown .Articles on Routers[M].Hephaestus Books ，2011：45-59.

[3] Andrew .Network Security Hacks Lockhart[M].USA :Media Inc ，2006：:245-230.

[4] Adrian Mackenzie .Wireless in Network[M].USA:Adrian MIT Press，2010：84-93

[5] Paul P .Linksys WRT54G Ultimate[M].USA:Larry Press.2007：145-149.

[6] Lime J.Articles on Embedded Linux Distributions[M].Hephaestus Books ，2011：96-103.

[7] Keith W.Ross .Lua’s Language [J].USA:2003：15-16.

[8] 杭州华三通信技术有限公司，路由交换技术(第 1 卷)[M] .北京： 清华大出版社，2011：87-89.

[9] Lambert M. OpenWrt Surhone[M]. Beta script Publishing，2011：168-171.

[10] John D. Safety Critical System: Challenges and Directions[C]. Proceedings of the 24th International Conference on Software Engineering, 2002：547-550.

[11] 徐恪，吴建平，徐明伟.高等计算机网络与路由器技术[M].机械工业出版社， 2009：86-94.

[12] 赖特 ，史蒂文斯.嵌入式系统消息处理与应用原理 [M]. 陆雪莹，译.机械工业出版社， 2004：358-360.

[13] 多伊尔，卡罗尔. 路由器配置操作与应用原理 [M]. 葛建立，译.人民邮电出版社，2006：66-69.

[14] 斯托林斯 J.操作系统精髓与设计原理[M].陈向群，译.机械工业出版社，2010：206-212.

[15] Agarwal R, Bachauri A. Towards Software Test Data Generation Using Binary Particle Swarm Optimization[R]. In proceedings of the National Systems Conference. 2008：181-201.

[16] 宋俊德，战晓苏.3G 原理系统与应用[M].北京：国防工业出版社，2008：

22-23.

[17] Maniyeri J.A Linux based software router supporting QoS， [M]. Eighth IEEE International Symposium onComputers and Communication，2003：101-102.

[18] 王奎，基于 Linux 的嵌入式消息处理[M] 机械工业出版社，2011：115-118

[19] 樊滨温，崔志强.路由器软件系统设计[M]. 电子工业出版社，2007：114-146.

[20] 曹利峰，杜学绘.嵌入式系统设计原理[M]. 电子工业出版社，2010：45-47.

[21] Klaus Wehrle. Linux 内核中网络协议的设计与实现[M].卢祖英，译. 北京：清华大学出版社，2006：340-352.

[22] Tamanathan, K. Chang, R, Kapur, L. Girod, E. Kohler, and D. Estrin, Sympathy for the Sensor Network Debugger, ACM SenSys, November 2005.

[23] 张宏科.IP 路由原理与技术[M] 北京：清华大学出版社, 2000:20-22.

[24] 送宝华. Linux 设备驱动开发详解[M].北京：人民邮电出版社，2008：56-57.

[25] P. Srisuresh. RFC3022-Traditional IP Network Address Translator [EB/OL].[2013-01].

[26] 孙琼.嵌入式 linux 应用程序开发详解[M].北京：人民邮电出版社, 2006:121-123.

[27] 詹荣开.嵌入式系统技术内幕与精髓[M].机械工业出版社，2012:98-99.

[28] 杨沙洲. Linux 环境下路由实现机制和扩展技术[M].人民邮电出版社，2012:126-128.

[29] Haghighat S, A Study of the Ground Illumination Footprint of Meteor Scatter Communication[J]. IEEE Trans Communication, 1996, 38(4)：50-52.

[30] 梁广民，王隆杰.思科网络实验室 CCNP(路由技术)实验指南[M]. 电子工业出版社，2012：289-293.

[31] Peter J. Integrating Flexible Support for Security Policies into the Linux Operating System[R], NSA and NAI labs, 2001：27-29.

[32] Baghighat A, A Study of the Ground Illumination Footprint of Meteor Scatter Communication[J]. IEEE Trans Communication, 1996, 38(4)：3-4.

[33] Gamamtitham K, Arya K. System Support for Embedded Applications[R]. 16th Internation Conference on VLSI Design. India, Indian institute of Technology. 2003：47-51.

[34] Borten Proschowsky. An intuitive text input method for touch wheels[R]. Proceedings of the SIGCHI conference on Human Factors in computing systems. New York, Technical University of Denmark. 2006：147-156.

[35] Hamma B,Helm H. Design Patterns: Elements of Reusable Object-Oriented Software[M]. Oxford University Press, 1997：65-66.

[36] 卡萨德 J.现代嵌入式操作系统原理[M].巩亚萍，译.人民邮电出版社 ，2012：56-58.

[37] Solastre, J. Gill and D. Culler, Versatile Low Power Media Access for Wireless Sensor Networks, ACM SenSys, November 2004.

[38] Sasilva J.S. European Third-Generation Mobile System[J]. IEEE Communication. Mag, 1996：83-84.

[39] 李勇.CGI 在嵌入式 WEB 服务器中的应用和实现[M]．机械工业出版社，2011：110-112.

[40] Hercauteren T, Bill Lin. Hardware/Software Communication and System Integration for Embedded Architectures[J]. Design Automation for Embedded Systems, 1997, 2：23-25.

[41] 吴朝晖.嵌入式软件设计与研发[M].电子工业出版社，2011：105-106.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于 OpenWrt 平台的路由器软件系统的设计优化与实现》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：张一多 日期：2013 年 7 月 4 日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：张一多 日期：2013 年 7 月 4 日

导师签名：宋彩霞 日期：2013 年 7 月 4 日

致　谢

论文撰写工作得以完成，首先要感谢我的校内导师宋颖慧副教授，从论文开题开始到论文撰写结束，宋老师始终给予无私的帮助与关心。在校外实习的一年中，宋老师给予的不仅是技术上的答疑和论文撰写的指导，而且在实习工作中，宋老师一直鼓励与教诲我如何完成项目组安排的技术工作。尤其是宋老师具有丰富的教学指导经验，这对我撰写并完成硕士学位论文起到了非常重要的帮助。在此，我对宋颖慧副教授表示衷心的感谢。

我将诚挚的感谢我的企业导师东软集团大连嵌入式事业部高级工程师郑淇文，我之所以能够完成项目上的工作这和郑淇文工程师密不可分。每当我遇到技术难题时，郑淇文工程师都能够以平易近人的态度，耐心为我解答。他认真的工作态度也给我留下了深刻的印象。同时，感谢部门领导田野、于海涛、位宏卓工程师以及项目组的每一位同事们，感谢他们的辛勤努力以及在工作中对我的关怀与鼓励，在我遇到困难时无私的帮助我克服困难，使我的论文工作能够顺利的完成。

感谢哈工大软件学院黄虎杰副院长和其他评审专家们，自从开题答辩和中期答辩，黄虎杰副院长亲自到大连为在大连实习的学生们组织答辩工作。在这里，我真挚的表达谢意，感谢专家老师们的不辞辛苦。

最后，我要感谢我的父母和我的家人。感谢父母和家人对我在学业上的支持。我将继续努力，来回报父母和家人对我的爱。在这里，我深深的说一声谢谢。

个人简历

2005 年 9 月于东北石油大学计算机科学学院攻读计算机科学与技术专业，2009 年 7 月毕业获工学学士学位。

2009 年 7 月于中国航天科工集团第三研究院 254 厂任信息化办公室科员。

2011 年 9 月考入哈尔滨工业大学软件学院攻读软件工程硕士。

2012 年 7 月开始在东软集团大连有限公司实习，参与 OpenWrt 平台下路由器研发项目，完成基于 OpenWrt 平台路由器软件系统的设计优化与实现。

基于OpenWrt平台的路由器软件系统的设计优化与实现

作者：张一弓
学位授予单位：哈尔滨工业大学

万方数据 WANFANG DATA 文献链接

引用本文格式：张一弓 基于OpenWrt平台的路由器软件系统的设计优化与实现[学位论文]硕士 2013