基于 WEB 的嵌入式监控系统

之

详细设计

Dev_Team	西邮 linux 兴趣小组嵌入式 Linux 组			
Change	许振文			
Members	许振文,武婷婷，贾二群			
M_num	M_date	M_version	M_part	M_reason
001	2008-12-06	0.01	所有	创建文档
002	2009-02-13	0.02	第 4 章	完善内容
003	2009-03-10	0.03	第 3，4 章	修改设计和 cgi 相关设计
004	2009-03-17	0.04	第 9 章	增加 SNMP 扩展一章

注:

M_num: 修改编号，从 **001** 开始，执行“**+1**”操作

M_dat: 修改日期

M_version: 修改后版本

M_part: 修改了那部分

M_reason: 修改说明

目 录

第1章 导言

1.1 目的

该文档是对《基于WEB的嵌入式监控系统》的概要设计，针对每个模块的具体功能需求，从技术的角度给出设计思路和解决方案。为《基于WEB的嵌入式监控系统》的软件开发人员提供指导参考资料。

1.2 范围

该文档是基于开发人员对该系统行业业务需求及初步调查写出的，目的为客户提供完整的业务需求，为开发人员提供系统开发基础要求。

本文档的预期读者是：

1） 设计人员

2） 开发人员

3） 测试人员

1.3 缩写说明

1） ARM：一种嵌入式处理器指令集,这里代指具有这种指令集的处理器。

2） Server：在嵌入式设备上提供监控服务的程序和进程。

3） Client：用户浏览器。

4） HttpServer：运行在嵌入式设备上和提供WEB服务。

5） OS：操作系统。

1.4 参考资料

1. 超文本传输协议-HTTP1.1 RFC文档（中文，英文）
2. 软件文档国家标准(GB8567-88)
3. 属于西邮linux兴趣小组已发表的其它文件

第2章 系统架构

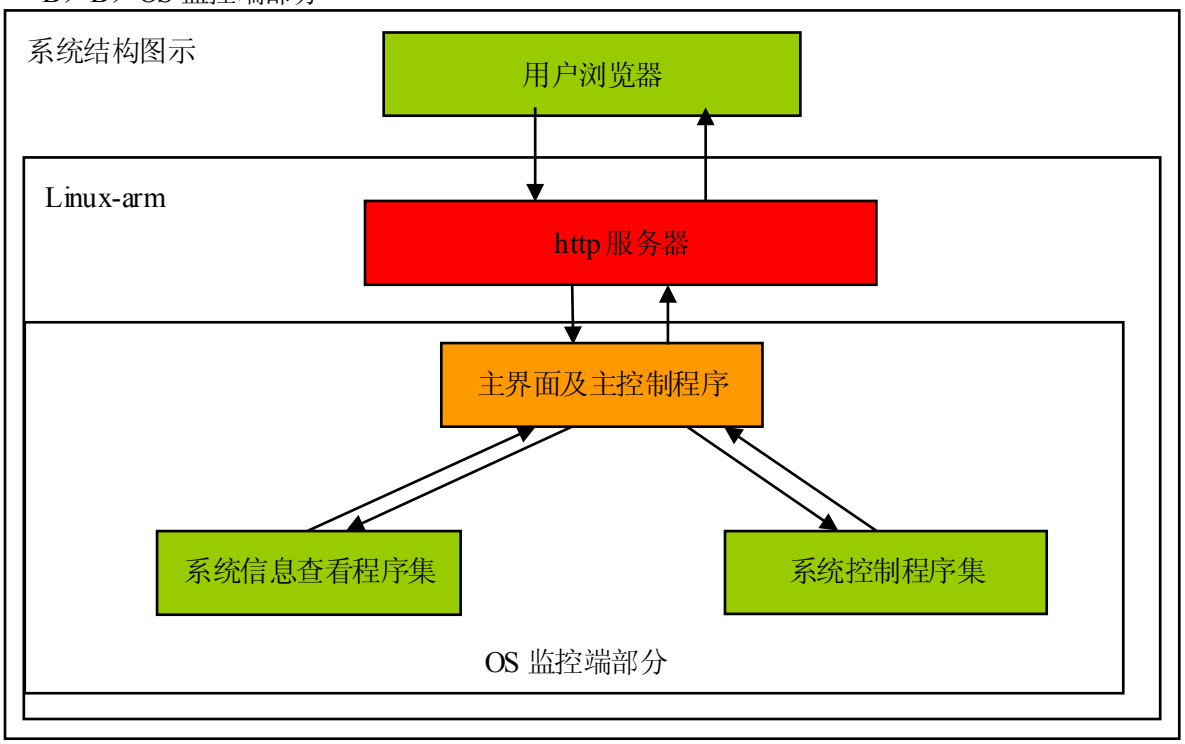
2.1 系统架构设计

程序结构为B/S 模式，其中运行在开发主机上的是浏览器，命名为Client，运行在ARM开发板上的是服务器端，命名为 Server。所以整个监控系统的重点在于服务器端的开发。

整个系统服务器端的设计分为两部分：

A）HttpServer 部分

B）B）OS 监控端部分

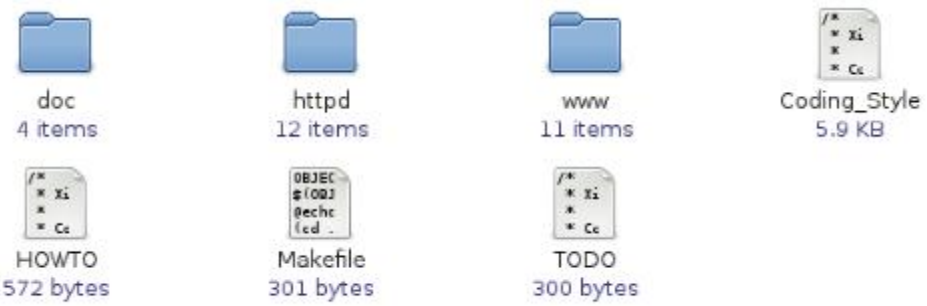


系统结构图示（图2-1）

2.2 系统文件组织

2.2.1 xhttp一根文件夹

```
helight@helight:xhttpd$ ls -l
total 32
-rw-r--r--  1 helight helight 6076 2008-12-14 09:55 Coding_Style
drwxr-xr-x  3 helight helight 4096 2008-12-14 10:01 doc
-rw-r--r--  1 helight helight  572 2008-12-14 09:55 HOWTO
drwxr-xr-x  3 helight helight 4096 2008-12-16 11:31 httpd
-rw-r--r--  1 helight helight  301 2008-12-14 09:55 Makefile
-rw-r--r--  1 helight helight  300 2008-12-14 09:55 TODO
drwxr-xr-x 12 helight helight 4096 2008-12-14 10:01 www
```



开发目录图示（图 2-2）

2.2.2 Httpd—服务器源文件

```
helight@helight:xhttpd$ ls httpd/ -l
total 100
-rw-r--r-- 1 helight helight 6568 2009-03-09 16:28 config.c
-rw-r--r-- 1 helight helight  847 2009-03-07 21:21 config.h
-rw-r--r-- 1 helight helight  917 2009-03-07 21:21 debug.h
-rw-r--r-- 1 helight helight 8547 2009-03-09 17:14 do_cgi.c
-rw-r--r-- 1 helight helight 2110 2009-03-09 16:19 do_cgi.h
-rw-r--r-- 1 helight helight 1244 2009-03-09 16:51 do_log.c
-rw-r--r-- 1 helight helight  347 2009-03-07 21:21 funs.c
-rw-r--r-- 1 helight helight  347 2009-03-07 21:21 funs.h
-rw-r--r-- 1 helight helight 1211 2009-03-07 21:21 http.h
-rw-r--r-- 1 helight helight 1833 2009-03-09 16:28 init.c
-rw-r--r-- 1 helight helight  410 2009-03-07 21:21 init.h
-rw-r--r-- 1 helight helight  433 2009-03-07 21:21 Makefile
-rw-r--r-- 1 helight helight 4076 2009-03-09 17:08 request.c
-rw-r--r-- 1 helight helight  438 2009-03-07 21:21 request.h
-rw-r--r-- 1 helight helight 4841 2009-03-09 16:28 response.c
-rw-r--r-- 1 helight helight  472 2009-03-07 21:21 response.h
-rw-r--r-- 1 helight helight 5721 2009-03-09 16:27 set_socket.c
-rw-r--r-- 1 helight helight  463 2009-03-07 21:21 set_socket.h
-rw-r--r-- 1 helight helight 1703 2009-03-09 16:26 xhttpd.c
-rw-r--r-- 1 helight helight 3206 2009-03-09 16:26 xhttpd.h
helight@helight:httpd$
```

2.2.3 www—应用程序源文件

```
helight@helight:xhttpd$ ls www/ -l
total 40
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 dev
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 disk
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 env
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 fs
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 img
-rw-r--r-- 1 helight helight 3770 2008-12-14 11:05 index.htm
```

```
-rw-r--r-- 1 helight helight    0 2008-12-14 09:55 Makefile
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 net
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 ps
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 sys
drwxr-xr-x 3 helight helight 4096 2008-12-14 10:01 usr
helight@helight:xhttpd$
```

2.2.4 文件结构说明：

目录结构说明（表 2-1）

文件夹	内容	说明
doc	Requirements_Analysis.pdf	项目需求说明书
	Design_Draft.pdf	项目概要设计文档
	Detail_Design.pdf	项目详细设计文档
	xhttpd_intor.pdf	项目介绍说明手册
httpd	Makefile	src 中的 Makefile 文件
	xhttpd.c	主文件，完成进程的初始化
	xhttpd.h	主头文件
	init.c	对服务器进行初始化
	init.h	
	config.c	配置文件读取并进行配置
	config.h	
	set_socket.c	建立服务器端监听接口
	set_socket.h	
	request.c	参数解析
	request.h	
	response.c	http 协议处理和 CGI 程序处理
	response.h	
	do_log.c	日志处理
	do_log.h	日志处理头文件
	debug.h	调试处理函数
	error.c	错误处理
	error.h	错误处理头文件
	funs.c	一般功能函数
	funs.h	

www	img	文件夹，存放网页图片，css，js 文件
	index.htm	网页主索引文件
	dev	设备管理应用程序
	disk	磁盘管理应用程序
	env	系统环境管理应用程序
	fs	文件系统管理应用程序
	net	网络管理应用程序
	ps	进程管理应用程序
	sys	系统服务管理应用程序
	usr	系统用户管理应用程序
make.conf	make.conf	Make 配置文件
Makefile	Makefile	主 Makefile 文件
Coding_Style	Coding_Style	代码编写规范
HOWTO	HOWTO	用户使用手册
TODO	TODO	开发指导手册

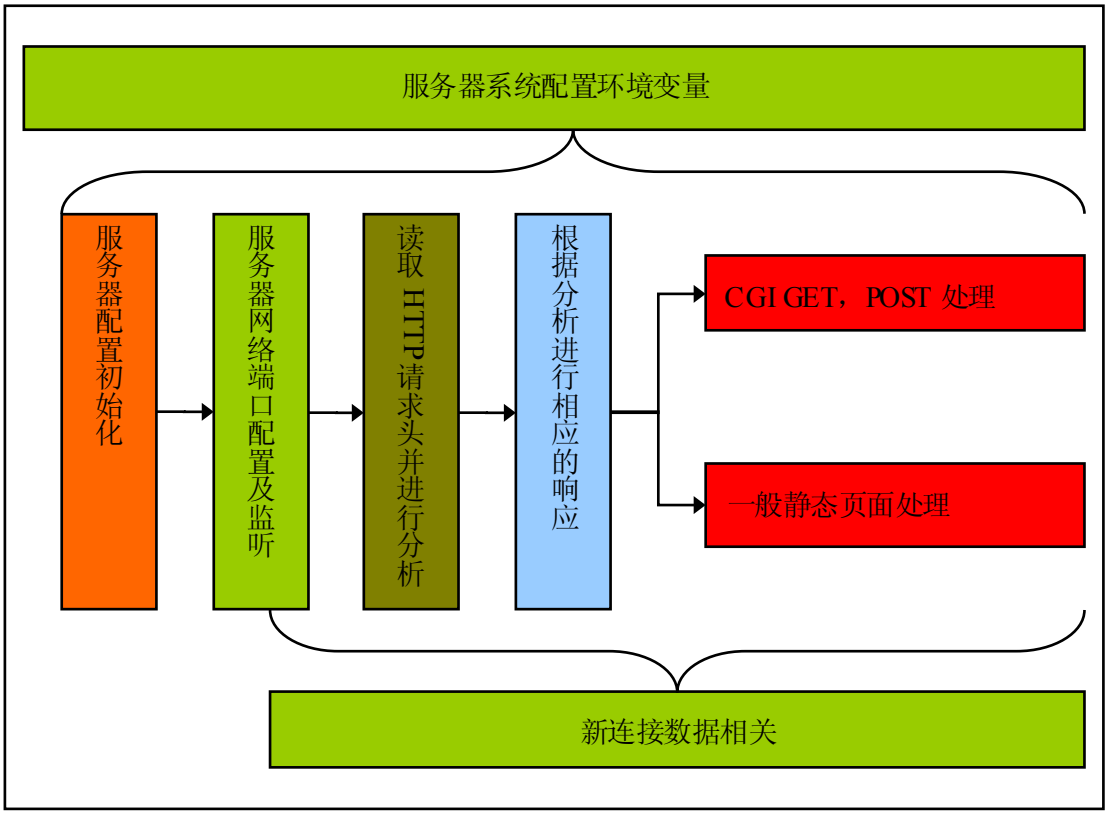
第3章 HttpServer 部分设计

3.1 系统功能模块划分

该部分作为整个系统的核心部分，负责系统的整体调度，从上面的系统结构图中可以看出，这一部分是整个系统的中心调度部分。负责 HTTP 协议的解析和处理，会话管理，并且调用监控程序完成系统的监控。是和用户浏览器直接交互的一块。

在设计实现上可划分为一下几个模块：

1. 系统配置初始化模块
2. 系统服务连接初始化模块及监听/接受服务模块
3. HTTP 协议头处理模块
4. 协议头处理模块
5. 静态页面处理模块
6. CGI POST，GET 请求处理模块



3.2 系统配置初始化模块

3.2.1 模块描述

对系统进行环境初始化和配置。

具体功能分为：

1) 读取配置文件

2) 根据配置文件完成相关系统环境变量的初始化。

3.2.2 数据相关

相关文件：config.c，config.h，xhttpd.h：

表3－1

文件	位置	描述
config.c	xhttpd/httpd/config.c	配置文件主程序
config.h	xhttpd/httpd/config.h	配置文件主程序头文件
xhttp.h	xhttpd/httpd/xhttpd.h	相关全局配置数据结构头文件

相关函数和数据：

xttp.h

```
/*
 * the config environment of the server,
 * It should be readed  from the configure file.
 *
 * @ port : listen port
 * @ start_time : when the server start
 * @ connection_timeout : how long should the connection keep when it's idle
 * @ max_connect : the max number client connect the server
 * @ log_file : the path of log file
 * @ version : the version of my server
 * @ name : the name of my server
 * @ index : the file name of the default
 * @ author : the name of the author
 * @ root_dir : the default server dir
 */

struct server_env{
        unsigned short  port;                   /*listen port*/
        unsigned int    start_time;             /**/
        unsigned int    connection_timeout;     /**/
        unsigned int    max_connect;            /*max number of connected! */
        char            *log_file;              /*path of log*/
        char            version[8];
```

```
        char            name[16];
        char            index[16];                  /*default index file*/
        char            author[32];                 /*the name of author*/
        char            root_dir[NAME_LEN];         /*document root*/
}server_env;

/*
 * the environment of the server when it is running ,
 * The date should be collect when the serve is running
 *
 * @connect : how many client connect to the server now
 * @ state : the state of my server -- r or d
 */

struct run_env{
    unsigned int    connect;                /*number of connected*/
    char        state;
}run_env;
```

config.h

```
#define  CONFIG_FILE    "config/xhttp_config"
#define  DEFAULT_DOC    "index.htm"     //#define  WEB_ROOT
"/home/helight/workspace/xhttpd/web/"
#define  WEB_ROOT       "www/"
#define  PORT           8080

void do_config(struct server_env *server_env);
void read_config(struct server_env *server_env);
void do_default_config(struct server_env *server_env);
```

3.2.3 算法

简单处理流程图：

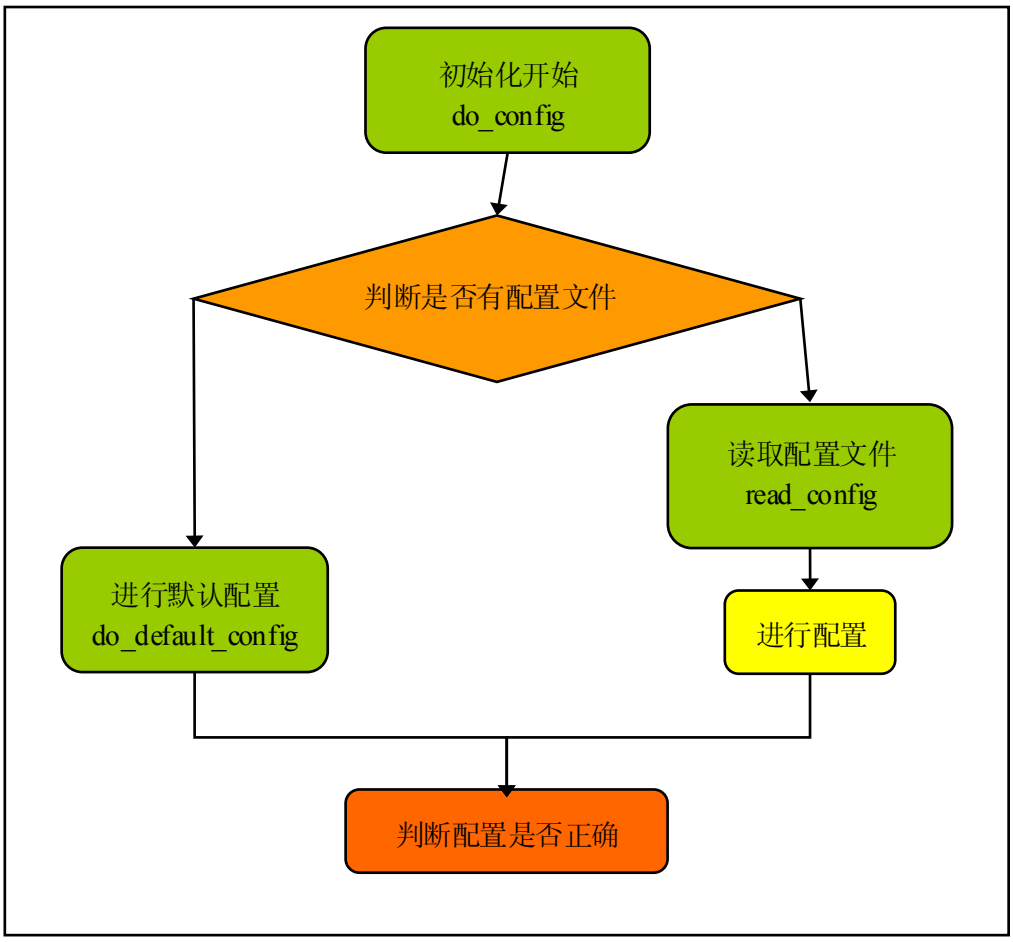


图3-1

算法描述：

3.2.4 接口

Config.h

```
void do_config(struct server_env *server_env);
void read_config(struct server_env *server_env);
void do_default_config(struct server_env *server_env);
```

Config.c

```
/*
 * do_config : do the config work after read config file or
 * do the default config
 *
 * @ *server_env : the point of server_env that used in all files
 *
 */

void do_config(struct server_env *server_env)
{
        struct stat statbuf;
```

```
        if(stat(CONFIG_FILE,&statbuf) == 0) {
                debug_where();
                read_config(server_env);
        } else {
                debug_where();
                do_default_config(server_env);
        }
}
/*
 *read_config : read config file
 *
 *@ *server_env : the point of server_env that used in all files
 *return :
 */
void read_config(struct server_env *server_env)
{
        return;
}
/*
 *do_default_config : do the default config
 *
 *@ *server_env : the point of server_env that used in all files
 *
 */
void do_default_config(struct server_env *server_env)
{
        char *tmp = (char *)WEB_ROOT;
        strncpy(server_env->root_dir, tmp, strlen(tmp));
        tmp = (char *)DEFAULT_DOC;
        strncpy(server_env->index, tmp, strlen(tmp));
        debug_print("http_dir: %s    index_file: %s\n",
                        server_env->root_dir, server_env->index);
        server_env->port = PORT;

        return;
}
```

3.2.5 使用约束

3.2.6 配置文件约束

配置文件中的有效配置项：

1. 端口 －－ port
2. 链接超时 －－ connection_timeout

3. 最大链接数 －－ max_connect
4. 主索引文件 －－ index
5. 主服务目录 －－ root_dir

件中的注释符号：#(英文井字符) ，如下示例

config the port

配置文件格式：name+spease+value+enter(配置名称＋空格＋值+换行)，如下示例

config the port

port 8080

3.2.7 测试相关

对默认配置进行和有配置文件都要进行测试。该部分只要进行功能测试，只要将配置文件中的相关数据写到服务器配置数据结构即可。

3.3 系统服务连接初始化及监听/接受服务模块

3.3.1 模块描述

该部分主要在于建立安全的socket服务，以便为下面的程序建立数据通道；并且接受请求，完成相应请求信息的分析。

具体功能：

1. 建立安全的 socket 服务。
2. 接受客户端请求
3. 建立相应的用户信息数据结构

3.3.2 数据相关

相关文件：set_socket.c，set_socket.h，xhttpd.h：

表3－2

文件	位置	描述
set_socket.c	xhttpd/httpd/set_socket.c	socket 建立实现的主程序
set_socket.h	xhttpd/httpd/set_socket.h	socket 建立实现的主程序头文件
xhttp.h	xhttpd/httpd/xhttpd.h	相关全局配置数据结构头文件

相关函数和数据：

httpd_listen(short port)：

```
int listenfd;
struct sockaddr_in  servaddr;
```

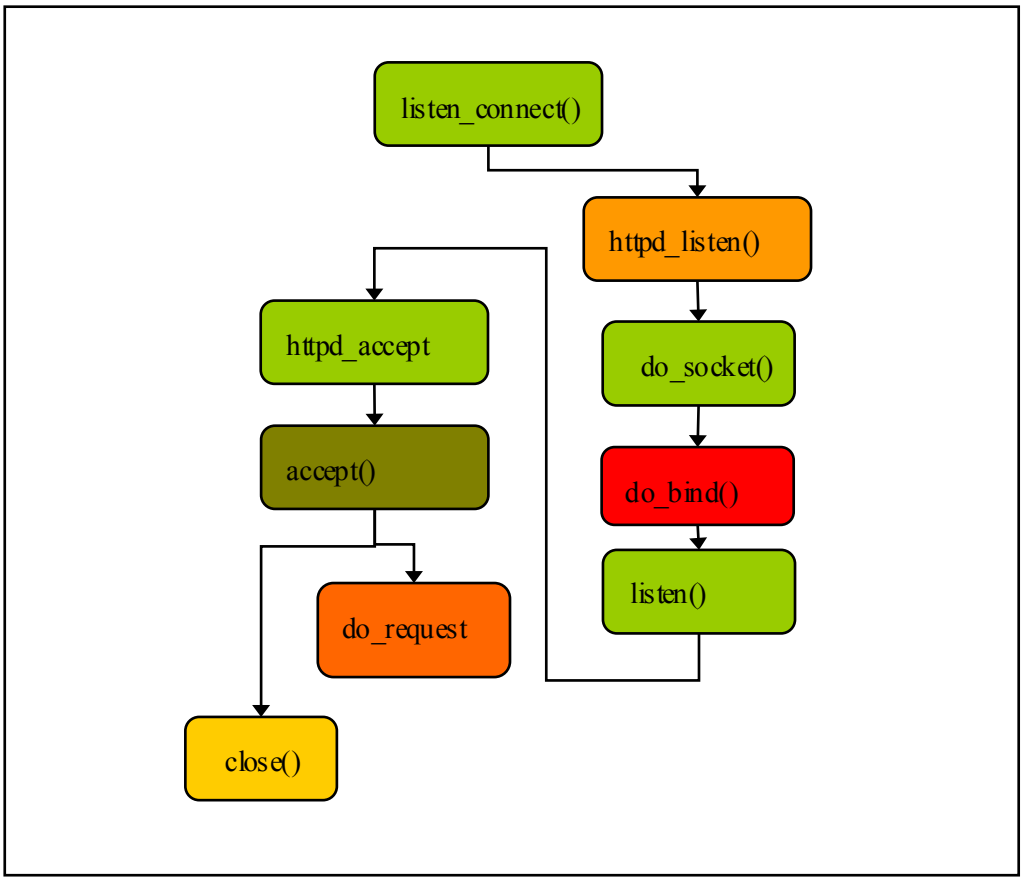
int httpd_accept(int listenfd)：

```
int connect_fd;
socklen_t len;
```

```
struct sockaddr_in client_addr;
```

3.3.3 算法

简单处理流程图：



3.3.4 接口

```
/*
 * listen_connect : the main module to setup the socket
 *
 * listen to connect
 */
void listen_connect(void);
/*
 * do_socket(void)
 * return fd of socket or -1 if fail
 *
 * deal with the error messages of socket
 */
static int do_socket(void)
/*
 * do_bind(int fd, sturct sockaddr *sadd, int len)
 * @ fd
```

```
 *
 * bind the ip address and the port
 */
static int do_bind(int fd, struct sockaddr *sadd, int len)
/*
 * http_listen(shor port)
 * @ port : the listen port on sercer
 *
 * set up the http listen
 */
int httpd_listen(short port)

/*
 * httpd_accept(int listenfd)
 * @ listenfd the file desc of connect socket
 *
 * accept the http request of brows
 */
int httpd_accept(int listenfd)
/*
 * do_close(int fd)
 * @ fd : fd
 * close the useless fd
 */
int do_close(int fd)
```

3.3.5 使用约束

该部分出了又主文件调用以外，其它函数均不可以调用。

3.3.6 测试相关

该部分的测试相对复杂，除了基本的功能测试意以外，更为重要的是性能测试。

3.4 HTTP协议头处理模块

3.4.1 模块描述

该模块完成处理HTTP 协议，主要完成对客户端发送的HTTP协议包进行解析。

3.4.2 功能

从客户端浏览器发送过来的请求数据中解析出其中的请求方式，请求文件和参数等。以便分析如何响应客户端的请求。

3.4.3 数据相关

相关文件：

相关文件：request.c，request.h：

表3—3

文件	位置	描述
request.c	xhttpd/httpd/request.c	头文件解析
request.h	xhttpd/httpd/request.h	相关结构申明

相关函数和数据：

```
/* The request method of brows
 * and now in this httpd we can do GET and POST
 */
enum method {
        GET,
        POST,
        UNKNOW,
};
/*
 * the state of the server
 */
enum state {
        C,      /*connect*/
        N,      /*not connnect*/
};
/*
 * version of http request
 * 1.0
 * 1.1
 */
enum httpv {
        V0,     /*version 1.0*/
        V1,     /*version 1.1*/
        UNV,
};
/*
 * for user info
 */
struct user_info{
        int             fd;             /*connect file descr*/
        unsigned short  port;           /*client port*/
        struct in_addr  ipaddr;         /*client ip*/
        unsigned long   login_time;     /*login time*/
        char            stat;
        enum httpv      httpv;          /*The request version of http */
```

```
    enum method     method;
    char            brows[16];
    char            filename[32];
    char            path[512];              /* the request path*/
    struct user_info *next;
};
```

HTTP 请求包头：

```
GET /demo/c.cgi HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.3) Gecko/2008092816 Iceweasel/3.0.3
(Debian-3.0.3-3)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://127.0.0.1:8080/
```

3.4.4 算法

简单处理流程图：

算法描述：

3.4.5 接口

该接口主要提供上层就收到连接请求后，进行 HTTP 协议头的分析操作。

```
void handle_request(struct user_info *new_user);
```

在该 c 文件中的其它函数都是静态的，不对外提供使用。

3.4.6 使用约束

该接口只有在 xhttp.c 中调用。在成功接受到请求连接后调用。

3.4.7 测试相关

该部分主要测试解析请求包头是否正确。主要使用 debug_print()来打印输出其中解析结果，并且与原信息包头作对比。

3.5 HTTP 处理模块

3.5.1 模块描述

该模块是根据已经分析后的数据作相应的处理。

3.5.2 功能

该模块是根据已经分析后的数据，寻找所请求的页面发送给浏览器，或是执行所请求的 CGI 程序进行运行并返回相应的数据，或是返回相应的错误信息。

3.5.3 数据相关

相关文件：

相关文件：response.c，response.h，http.h：

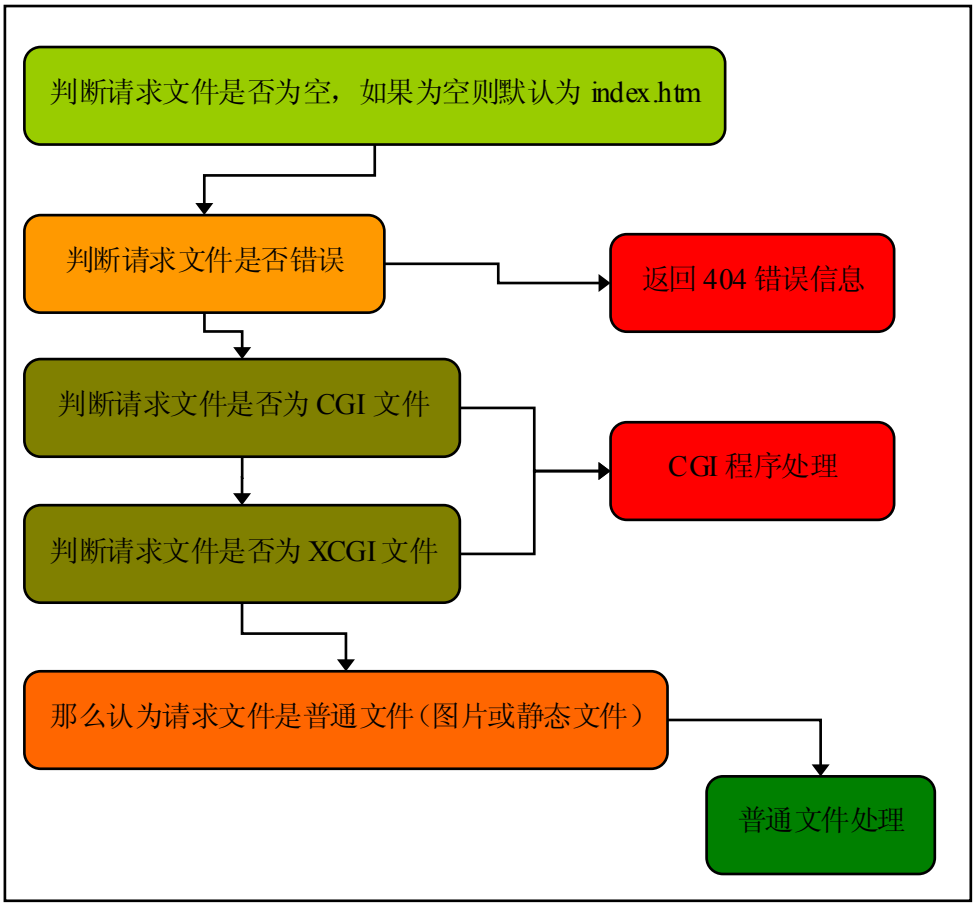
表3—5

文件	位置	描述
response.c	xhttpd/httpd/response.c	头文件解析
response.h	xhttpd/httpd/response.h	相关结构申明
http.h	xhttpd/httpd/http.h	HTTP 协议相关的定义和申明

相关函数和数据：

3.5.4 算法

简单处理简单处理流程图：



算法描述：

3.5.5 接口

```
int do_response(struct user_info *new_user);
int http_nak(int fd, int code, const char *nak);
```

3.5.6 使用约束

接口 int do_response(struct user_info *new_user) 只有在 request.c 文件中调用。其余文件中均不可调用。

接口 int http_nak(int fd, int code, const char *nak)可以在 request.c 和 do_cgi.c 中被调用。

3.5.7 测试相关

3.6 静态页面处理模块

3.6.1 模块描述

该部分主要处理浏览器对图片、css 文件等静态文件的请求。

3.6.2 功能

该部分主要处理浏览器对图片、css 文件等静态文件的请求。

3.6.3 数据相关

相关文件：

相关文件：response.c，response.h，http.h：

表3—6

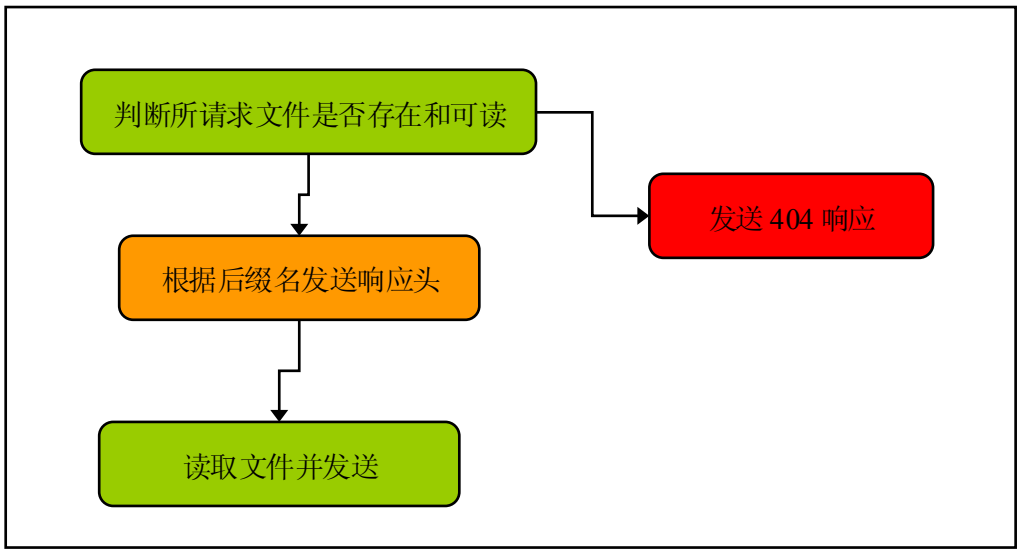
文件	位置	描述
response.c	xhttpd/httpd/response.c	头文件解析
response.h	xhttpd/httpd/response.h	相关结构申明
http.h	xhttpd/httpd/http.h	HTTP 协议相关的定义和申明

相关函数及数据：

```
static int send_head(const char *ack, int len, const char *path, int fd);
static int common_file(char *pathname, int fd);
```

3.6.4 算法

简单处理流程图：



算法描述：

3.6.5 接口

```
static int common_file(char *pathname, int fd);
```

3.6.6 使用约束

该接口由上一节中的 do_response 调用。

3.6.7 测试相关

3.7 CGI GET，POST 处理模块

3.7.1 模块描述

该部分完成 CGI 和 XCGI 程序的处理。

3.7.2 功能

该部分完成 CGI 和 XCGI 程序的处理。分别根据 GET，POST 和 XCGI，CGI 来分别处理。

3.7.3 数据相关

相关文件：

相关文件：do_cgi.c，do_cgi.h，http.h：

表3—6

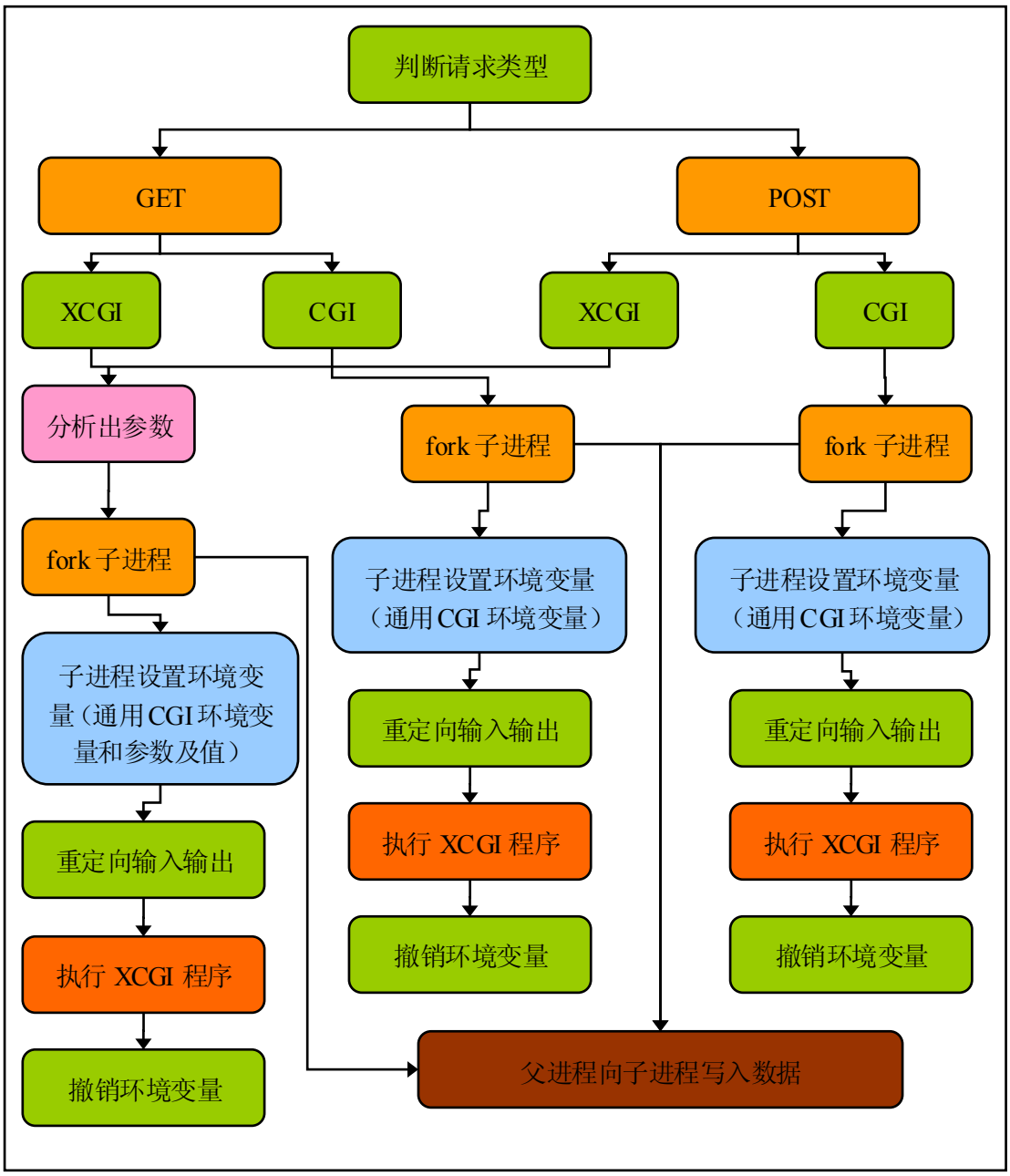
文件	位置	描述
do_cgi.c	xhttpd/httpd/do_cgi.c	头文件解析
do_cgi.h	xhttpd/httpd/do_cgi.h	相关结构申明
http.h	xhttpd/httpd/http.h	HTTP 协议相关的定义和申明

相关函数及数据：

```
int do_cgi(struct user_info *new_user);
static void response_xcgi_get(struct user_info *new_user);
static void response_cgi_get(struct user_info *new_user);
static void response_xcgi_post(struct user_info *new_user);
static void response_cgi_post(struct user_info *new_user);
static void decode(char *ch);
static void set_cgi_common_env(struct user_info *new_user);
static void unset_cgi_common_env(struct user_info *new_user);
static void send_cgi_head_response(enum httpv);
```

3.7.4 算法

简单处理流程图：



算法描述：

3.7.5 接口

```
int do_cgi(struct user_info *new_user);
```

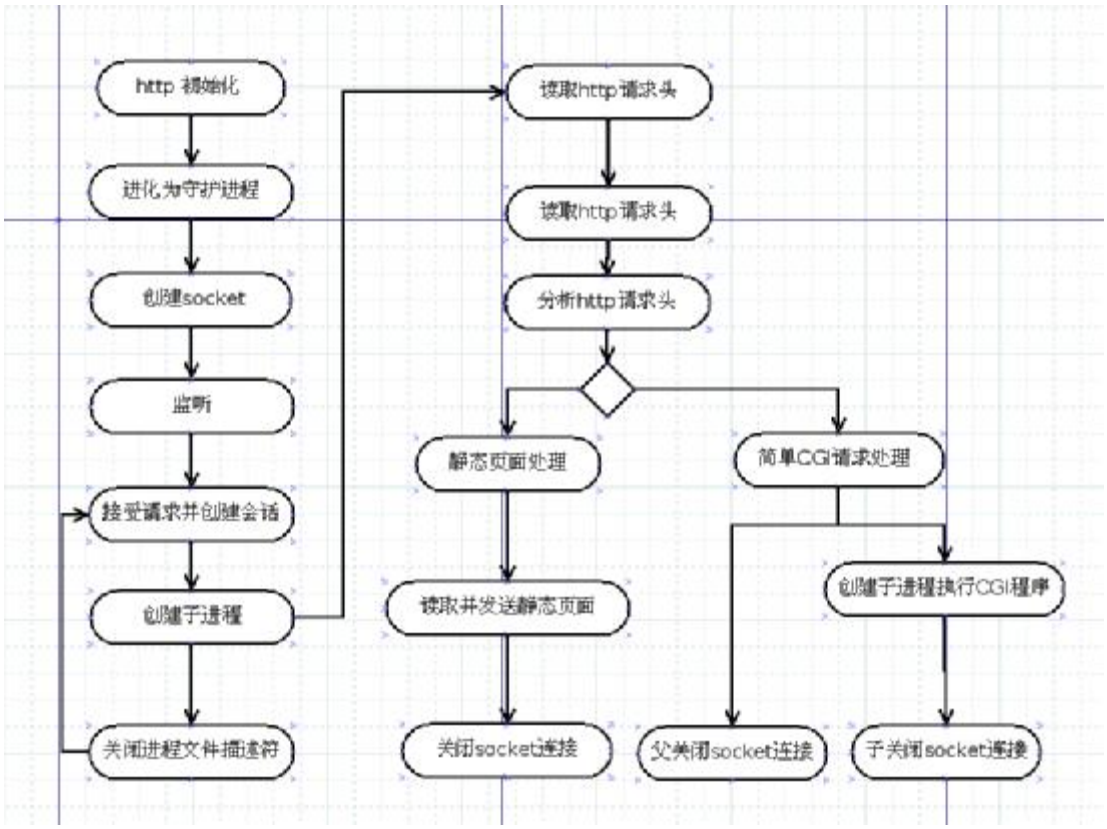
3.7.6 使用约束

该接口在 do_response 中被调用，其余地方均不可使用。

3.7.7 测试相关

3.8 Http 协议的设计实现

下面画出了 http 系统的内部调度过程。



HTTPD 系统的内部调度过程图（3-1）

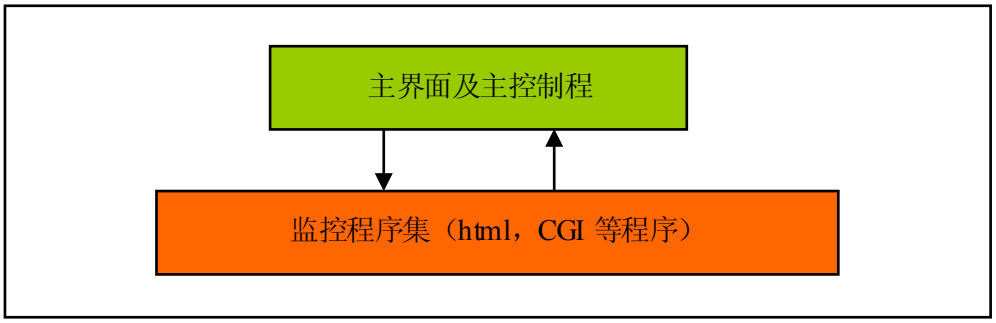
其中日志的记录贯穿于整个系统调度过程中，故没有单独列出来。

第4章 OS 监控端主控模块

该部分负责系统信息的收集和对系统实施控制，该部分程序现对比较庞大，也比较复杂。该部分正真实现系统信息的收集和系统控制命令的执行。是整个监控系统的最底层的部分，这部分相对分的比较细，根据具体功能实现的不同来写具体的模块

4.1 主控程序

根据“系统结构图示”，可以将系统 http 下面部分设计为可扩展的系统。让系统功能以插件的形式添加，并且开放系统插件开发协议与标准，最大限度的让更多的人来开发和丰富插件功能。



可扩展框架图示**(4-1)**

4.2 系统主控程序设计

4.2.1 模块描述

主要是按照要求和用户的不同生成相应的用户界面。用户分为普通用户和管理员用户。程序应该根据不同的用户生成不用的用户界面。

它本身并不完成下层功能文件的调用执行，下层功能文件的真正调用执行还是有HTTP服务器来调用触发的，它只是完成对下层功能的组织，能够在用户点击或是提交主界面上提供的功能时，能够给服务器发送相关的调用命令。从而服务器能够正确的调用执行下层功能文件。

4.2.2 功能

1） 完成系统主控程序环境变量的初始化，

2） 实现系统插件的查找和识别。

3） 实现已找到插件的分类，类别会在扩展框架标准中给出。

4） 根据插件分类和功能需求生成相应的用户界面。

4.2.3 数据相关

相关文件：

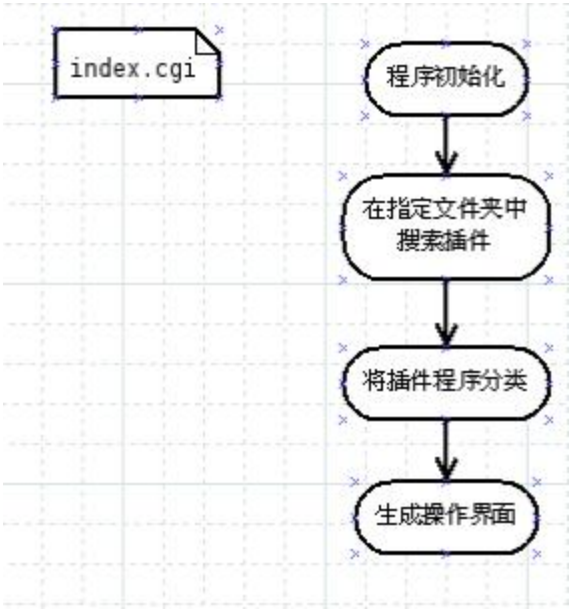
源文件文件	描述
Index.h	函数声明和相关数据结构定义
Index.c	用于生成主文件 index.cgi，
Iserch.h	函数声明和数据结构定义

Iserch.c	完成搜索和按权限和类别分类。

相关函数及数据：

4.2.4 算法

简单处理流程图：



算法描述：

1） 完成系统主控程序环境变量的初始化，

2） 将系统插件按照系统规则进行查找和识别。主要是在功能文件夹内部进行扫描。

3） 把已找到的到插件安系统规则进行分类，类别会在扩展框架标准中给出。

4） 根据插件分类和功能需求生成相应的用户界面。目前实现会在 js 代码的帮助之下，写到带编号的 div 中。

4.2.5 接口

HTTP 服务器对它的调用是文件的执行调用，而不是功能函数的调用，对HTTP 的返回数据是直接输出到它的标准输出中即可。

对于下层功能的调用也是以文件的调用来实现的，所以无需想下层提供接口。在这里它只是对下层功能文件的组织，而实际的调用还是在 HTTP 中请求触发的。

4.2.6 使用约束

4.2.7 测试相关

4.3 cgi程序开发标准

4.3.1 总体描述

该部分主要是规定在本项目中的嵌入式web服务器下CGI 程序的开发规范和标准。主要是方便CGI 程序开发者更好，更方便的开发应用程序。应用CGI 程序均在程序文件夹“www”目录中。该部分主要叙述表中CGI 程序的编程规范。

CGI规定了 Web 服务器调用其他可执行程序(CGI 程序)的接口协议标准。Web服务器通过调用CGI 程序实现和 Web浏览器的交互。CGI 程序可以用任何程序设计语言编写,如 Shell脚本语言、 Perl、Fortran、Pascal、C 语言等。但是用C 语言编写的CGI程序具有执行速度快、安全性高等特点。本小节主要分析用 C语言进行CGI 程序设计的方法、过程和技巧。

4.3.2 规范内容

CGI 接口标准包括标准输入、环境变量、标准输出三部分。

1） 标准输入

2） 环境变量

3） 标准输出

4.3.3 标准输入

CGI 程序像其他可执行程序一样,可通过标准输入(stdin)从 Web 服务器得到输入信息,如Form 中的数据,这就是所谓的向CGI 程序传递数据的 POST 方法。这意味着在操作系统命令行状态可执行CGI 程序,对CGI 程序进行调试。POST方法是常用的方法,本文将以此方法为例,分析CGI 程序设计的方法、过程和技巧。

4.3.4 环境变量

操作系统提供了许多环境变量,它们定义了程序的执行环境,应用程序可以存取它们。Web服务器和CGI接口又另外设置了自己的一些环境变量,用来向CGI 程序传递一些重要的参数。CGI 的GET 方法还通过 环境变量QUERY-STRING向CGI 程序传递 Form中的数据。

环境变量是文本串(名字/值对),可以被OS Shell 或其他程序设置 ,也可以被其他程序访问。它们是 Web 服务器传递数据给CGI 程序的简单手段,之所以称为环境变量是因为它们是全局变量,任何程序都可以存取它们。

下面是CGI 程序设计中常常要用到的一些环境变量。

HTTP-REFERER:调用该CGI 程序的网页的URL。

REMOTE-HOST:调用该CGI 程序的 Web 浏览器的机器名和域名。

REQUEST-METHOD:指的是当 Web 服务器传递数据给 CGI 程序时所采用的方法,分为GET 和 POST 两种方法。GET 方法仅通过环境变量(如 QUERY-STRING)传递数据给 CGI程序,而 POST 方法通过环境变量和标准输入传递数据给 CGI 程序,因此 POST方法可较方便地传递较多的数据给 CGI 程序。

SCRIPT-NAME:该CGI 程序的名称。

QUERY-STRING:当使用 POST 方法时,Form 中的数据最后放在 QUERY-STRING 中,传递给 CGI程序。

CONTENT-TYPE:传递给 CGI程序数据的 MIME类型,通常为"applica tion /x-www-form-url encodede",它是从 HTML Form中以 POST方法传递数据给 CGI 程序的数据编码类型,称为URL 编码类型。

CONTENT-LENGTH:传递给 CGI 程序的数据字符数(字节数)。

在 C 语言程序中,要访向环境变量,可使用 getenv()库函数。例如:

if (getenv ("CONTENT-LENGTH"))

n=atoi(getenv ("CONTENT-LENGTH"));

请注意程序中最好调用两次 getenv():第一次检查是否存在该环境变量,第二次再使用该环境变量。这是因为函数 getenv()在给定的环境变量名不存在时,返回一个 NULL(空)指针,如果你不首先检查而直接引用它,当该环境变量不存在时会引起 CGI 程序崩溃。

当用户提交一个 HTML Form 时,Web 浏览器首先对 Form 中的数据以名字/值对的形式进行编码,并发送给 Web 服务器,然后由 Web 服务器传递给 CGI 程序。其格式如下:

name1=value1&name2=value2&name3=value3&name4=value4&...

其中名字是 Form 中定义的 INPUT、SELECT 或 TEXTAREA 等标置(Tag)名字,值是用户输入或选择的标置值。这种格式即为 URL 编码,程序中需要对其进行分析和解码。要分析这种数据流,CGI 程序必须首先将数据流分解成一组组的名字/值对。这可以通过在输入流中查找下面的两个字符来完成。

每当找到字符=,标志着一个 Form 变量名字的结束;每当找到字符& ,标志着一个 Form 变量值的结束。请注意输入数据的最后一个变量的值不以&结束。

一旦名字/值对分解后,还必须将输入中的一些特殊字符转换成相应的 ASCII 字符。这些特殊字符是:

+:将+转换成空格符;

%xx:用其十六进制 ASCII 码值表示的特殊字符。根据值 xx 将其转换成相应的 ASCII 字符。

对 Form 变量名和变量值都要进行这种转换。

4.3.5 目前本系统实现的环境变量

目前已经实现的 CGI 环境变量有一些几个：

REQUEST-METHOD：GET 或是 POST

QUERY-STRING： GET 方式传递的参数

CONTENT-LENGTH：参数的总长度

HTTP_COOKIE： 设置的 cookie 值

其余正在进一步实现当中。。。。。

4.3.6 标准输出

CGI 程序通过标准输出(stdout)将输出信息传送给 Web 服务器。传送给 Web 服务器的信息可以用各种格式,通常是以纯文本或者 HTML 文本的形式,这样我们就可以在命令行状态调试 CGI 程序,并且得到它们的输出。

下面是一个简单的 CGI 程序,它将 HTML 中 Form 的信息直接输出到 Web 浏览器。

```
#include <stdio.h>
#include <stdib.h>
main()
```

```
{
   int,i,n;
printf ("Contenttype:text/html\r\n\r\n");
n=0;
if(getenv("CONTENT-LENGTH"))
n=atoi(getenv(CONTENT-LENGTH"));
for (i=0;i<n;i++)
putchar(getchar());
putchar ('n');
fflush(stdout);
}
```

下面对此程序作一下简要的分析。

```
prinft ("Contenttype:text/html\r\n\r\n");
```

此行通过标准输出将字符串"Contenttype:text/html\r\n\r\n"传送给 Web 服务器。它是一个 MIME 头信息,它告诉 Web 服务器随后的输出是以纯 ASCII 文本的形式。请注意在这个头信息中有两个新行符,这是因为 Web 服务器需要在实际的文本信息开始之前先看见一个空行。

```
if (getenv("CONTENT-LENGTH"))
n=atoi (getenv("CONTENT-LENGTH"));
```

此行首先检查环境变量 CONTENT-LENGTH 是否存在。Web 服务器在调用使用 POST 方法的 CGI 程序时设置此环境变量,它的文本值表示 Web 服务器传送给 CGI 程序的输入中的字符数目,因此我们使用函数atoi() 将此环境变量的值转换成整数,并赋给变量n。请注意 Web 服务器并不以文件结束符来终止它的输出,所以如果不检查环境变量 CONTENT-LENGTH, CGI 程序就无法知道什么时候输入结束了。

```
for (i=0;i<n;i++)
putchar(getchar());
```

此行从 0 循环到(CONTENT-LENGTH-1)次将标准输入中读到的每一个字符直接拷贝到标准输出,也就是将所有的输入以 ASCII 的形式回送给 Web 服务器。

通过此例,我们可将 CGI 程序的一般工作过程总结为如下几点。

1.通过检查环境变量 CONTENT-LENGTH,确定有多少输入;

2.循环使用 getchar()或者其他文件读函数得到所有的输入;

3.以相应的方法处理输入;

4.通过"Contenttype:"头信息,将输出信息的格式告诉 Web 服务器;

5.通过使用 printf()或者 putchar()或者其他的文件写函数,将输出传送给 Web 服务器。

总之,CGI 程序的主要任务就是从 Web 服务器得到输入信息,进行处理,然后将输出结果再送回给 Web 服务器。

4.4 xcgi 开发标准

该部分主要是规定在本项目中的嵌入式 web 服务器下 CGI 程序的开发规范和标准。主

要是方便CGI 程序开发者更好，更方便的开发应用程序。应用CGI 程序均在程序文件夹“www”目录中。

包括CGI 程序的命名规范，程序中变量定义规范和使用规范。

4.4.1 规范内容

4） CGI 程序的命名规范。

5） 变量定义规范。

6） GET 方式参数传递和使用规范。

7） POST 方式参数传递和使用规范。

4.4.2 CGI 程序命名规范标准

1） 应用程序的命名最好做到见名知意。

以下是”www“目录的结构:

```
helight@helight:xhttpd$ ls www/
dev  disk  env  fs  img  index.htm  Makefile  net  ps  sys  usr
helight@helight:xhttpd$
```

加粗者均是文件夹，在个个文件夹下面存放具体的CGI 应用程序。例如文件夹“fs”下存放的就是关于文件系统管理的CGI 应用程序。

2） 根据CGI 应用程序的功能进行分类，将具有一类功能或是对同一对象进行管理的应用程序保存到同一文件夹中。

4.4.3 变量定义规范

变量的定义为了统一和防止变量混淆，在本系统中的变量名统一使用相同的前缀：XY_。

4.4.4 GET 方式参数传递和使用规范标准

1） 变量的定义为了统一和防止变量混淆，在本系统中的变量名统一使用相同的前缀：XY_

2） 传递的参数个数不多于 5 个

3） 传递参数的变量名长度不多于 32

4） 变量参数值的长度不多于 64

4.4.5 POST 方式参数传递和使用规范标准

1） 变量的定义为了统一和防止变量混淆，在本系统中的变量名统一使用相同的前缀：XY_

2） 传递的参数个数不多于 5 个

3） 传递参数的变量名长度不多于 32

4） 变量参数值的长度不多于 64

4.4.6 目前本系统实现的环境变量

目前已经实现的CGI 环境变量有一些几个：

REQUEST-METHOD：GET 或是 POST

QUERY-STRING： GET 方式传递的参数

CONTENT-LENGTH：参数的总长度

HTTP_COOKIE： 设置的 cookie 值

其余正在进一步实现当中。。。。。

4.4.7 Shell XCGI 编写规范

4.4.8 C XCGI 程序编写规范

4.5 XCGI 程序 Demo

本 demo 程序是用于示范如何编写在该系统下的 XCGI 应用程序。目前 demo 程序的调试是在主页面文件管理一栏中，目前有 shell 写的 XCGI 的示例程序和 C 语言写的一个用户 POST 方法的测试程序。

4.5.1 shell 程序：

该程序是 shell 程序通过 get 传递参数的例子：

```
#!/bin/sh
#echo "Content-type:text/html\r\n\r\n"
echo "<Cotent-type: text/html>"
echo
echo "<html><head><title>The content you want</title>"
echo "<meta http-equiv="Content-Style-Type"content="text/css";charset="gb2312">"
echo "</head><body  background='./img/boo.jpg'; bgcolor='#FFFFFF'>"

echo "<h1 align=center style="border-style:outset">The imformation you need is</h1>"
echo "<div style='margin:48em'></div>"
echo "<hr>"

echo $XY_path
if [ -z $XY_path ]
    then
    XY_path=xu_zhenwen
fi

echo "<span style='width:40px;height:50px;'><a
href='shell.cgi?XY_path=$XY_path$XY_path'>$XY_path</a></span>"

echo "<hr>"
echo "<a
href='shell.cgi?XY_path=home&XY_name=zhenwen&XY_www=zhwen.org&XY_dim4=dim4.cn&XY_gmail=h
elight&XY_xux=xux'>test</a>"
```

```
echo "<hr>"
echo "<hr></body></html>"
```

4.5.2 C 程序：

该程序用于获取 cookie 的值和以 POST 方式发送的表单数据：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
        char buf[1024];
        char *cookie = getenv("HTTP_COOKIE");           //获取 cookie 的值
        int len = atoi(getenv("CONTENT_LENGTH"));       //获取数据长度
        printf("Set-Cookie:user=helight.xu\r\n");
        printf("Content-type:text/html\r\n\r\n");
        printf("<html><head><title>cgi post</title>\
                        <meta http-equiv='Content-Type' content='text/html; charset=utf-8' />");

        printf("<body><div width=900px><div align=center>\
                        <h1>cgi post</h1></div><hr> <div>");

        read(0, buf, len);                              //读取数据
        buf[len] = '\0';
        printf("buf:%s\n", buf);                        //显示数据
        printf("cookie:%s len:%d\n", cookie, len);      //显示 cookie 中的值

         printf("<form action=spost.cgi method=post>");
        printf("<input type=text name=XY_wd   size=42 maxlength=100><br>");
        printf("<input type=text name=XY_ww   size=42 maxlength=100><br>");

       printf("<br><input type=submit value=配置 >\
                        <input type=reset value=重置 ></form>\
                        </div><hr></div></body></html>\n");

        return 0;
}
```

4.6 CGI 参数解析程序－cgilib

4.6.1 模块描述

由于 CGI 应用程序的参数交互是通过环境变量或是读写管道来实现的，而且在参数传递到 CGI 程序时并没有进行解析。所以在 CGI 应用程序端一个首要解决的问题就是参数的

解析。由于该解析对所有 CGI 应用程序都可以应用，所以就需要写成一个公用模块。

4.6.2 功能描述

主要就是将环境变量中获取的参数进行解析，解析为相应的参数值以供应用程序使用。

4.6.3 数据相关

相关文件：cgilib.c, cgilib.h

表4—2

文件	位置	描述
cgilib.c	xhttpd/www/cgilib.c	解析函数实现的主程序
cgilib.h	xhttpd/www/cgilib.h	提供的接口头文件

相关函数及数据：

4.6.4 算法

简单处理流程图：

算法描述：

4.6.5 接口

4.6.6 使用约束

4.6.7 测试相关

第5章 系统信息查看部分

主要是完成对系统信息的收集，并将其返回给 HttpServer，再由 HttpServer 发送到请求的浏览器端。对应系统功能定义中的《系统信息浏览功能》部分。(见需求分析之 2.2-A）

根据查看信息的不同分为如下几个方面的设计：

5.1 系统文件信息的查看

5.1.1 模块描述

5.1.2 数据相关

相关文件：

相关函数及数据：

5.1.3 算法

简单处理流程图：

算法描述：

5.1.4 接口

5.1.5 使用约束

5.1.6 测试相关

5.2 系统磁盘信息的查看

5.2.1 模块描述

5.2.2 数据相关

相关文件：

相关函数及数据：

5.2.3 算法

简单处理流程图：

算法描述：

5.2.4 接口

5.2.5 使用约束

5.2.6 测试相关

5.3 系统进程信息的浏览

5.3.1 模块描述

5.3.2 数据相关

相关文件：

相关函数及数据：

5.3.3 算法

简单处理流程图：

算法描述：

5.3.4 接口

5.3.5 使用约束

5.3.6 测试相关

5.4 系统网络配置的查看

5.4.1 模块描述

5.4.2 数据相关

相关文件：

相关函数及数据：

5.4.3 算法

简单处理流程图：

算法描述：

5.4.4 接口

5.4.5 使用约束

5.4.6 测试相关

5.5 系统用户信息的查看

5.5.1 模块描述

5.5.2 数据相关

相关文件：

相关函数及数据：

5.5.3 算法

简单处理流程图：

算法描述：

5.5.4 接口

5.5.5 使用约束

5.5.6 测试相关

5.6 溶合 powertop 功能

5.6.1 模块描述

5.6.2 数据相关

相关文件：

相关函数及数据：

5.6.3 算法

简单处理流程图：

算法描述：

5.6.4 接口

5.6.5 使用约束

5.6.6 测试相关

5.7 系统其它情况的查看

5.7.1 模块描述

5.7.2 数据相关

相关文件：

相关函数及数据：

5.7.3 算法

简单处理流程图：

算法描述：

5.7.4 接口

5.7.5 使用约束

5.7.6 测试相关

第6章 系统控制部分

该部分主要完成系统控制命令的执行，HttpServer 在接受浏览器端有效控制命令之后，HttpServer会调用该部分相应的控制模块去执行相应的控制操作。对应系统功能定义中的《系统控制功能》部分。(见需求分析之 2.2-B)

6.1 系统用户的管理

6.1.1 模块描述

6.1.2 数据相关

相关文件：

相关函数及数据：

6.1.3 算法

简单处理流程图：

算法描述：

6.1.4 接口

6.1.5 使用约束

6.1.6 测试相关

6.2 系统进程的管理

6.2.1 模块描述

6.2.2 数据相关

相关文件：

相关函数及数据：

6.2.3 算法

简单处理流程图：

算法描述：

6.2.4 接口

6.2.5 使用约束

6.2.6 测试相关

6.3 系统文件系统的管理

6.3.1 模块描述

6.3.2 数据相关

相关文件：
相关函数及数据：

6.3.3 算法

简单处理流程图：

算法描述：

6.3.4 接口

6.3.5 使用约束

6.3.6 测试相关

6.4 其它系统设备管理

6.4.1 模块描述

6.4.2 数据相关

相关文件：

相关函数及数据：

6.4.3 算法

简单处理流程图：

算法描述：

6.4.4 接口

6.4.5 使用约束

6.4.6 测试相关

6.5 系统的其它管理

6.5.1 模块描述

6.5.2 数据相关

相关文件：

相关函数及数据：

6.5.3 算法

简单处理流程图：

算法描述：

6.5.4 接口

6.5.5 使用约束

6.5.6 测试相关

第7章 公共模块设计

7.1 日志记录模块

7.1.1 模块描述

7.1.2 数据相关

相关文件：

相关函数及数据：

7.1.3 算法

简单处理流程图：

算法描述：

7.1.4 接口

7.1.5 使用约束

7.1.6 测试相关

7.2 调试模块

7.2.1 模块描述

7.2.2 数据相关

相关文件：

相关函数及数据：

7.2.3 算法

简单处理流程图：

算法描述：

7.2.4 接口

7.2.5 使用约束

7.2.6 测试相关

7.3 错误处理模块

7.3.1 模块描述

7.3.2 数据相关

相关文件：

相关函数及数据：

7.3.3 算法

简单处理流程图：

算法描述：

7.3.4 接口

7.3.5 使用约束

7.3.6 测试相关

第8章 用户界面部分

Client 端界面由于是浏览器，所以在服务端写程序时要求要写成以html 代码输出，再由浏览器解释为图形界面。

该部分是显示页面的部分，该部分完成系统界面的生成。使用C 程序或是 shell 程序编写，或者使用静态的 html编写，对应系统功能定义中的《提供比较友好的用户界面》部分(见2.2-C-1)。

主功能界面设计采用经典的”工“字形结构设计。如下所示：

<table>
<tr><td colspan="2">logo 图标</td></tr>
<tr><td colspan="2">banner 主菜单</td></tr>
<tr><td>二级菜单
显示主菜单
下的二级菜单</td><td>显示菜单项对应的系统信息
显示控制系统的具体操作面板</td></tr>
<tr><td colspan="2">版权信息</td></tr>
</table>

第9章 Net-snmp 接口实现

9.1 Net-snmp 介绍

Simple Network Management Protocol (SNMP) 是一个被广泛使用的协议，可以监控网络设备（比如路由器）、计算机设备甚至是 UPS。NET-SNMP 是一种开放源代码的简单网络管理协议（Simple Network Management Protocol）软件。Net-SNMP 是用于实施 SNMP v1，SNMP v2，SNMPv3 的应用程序套件，可以使用在 IPv4、IPv6 的环境中。这个套件包括：

命令行程序包括：

从支持 SNMP 的设备中检索信息的命令。用于执行单个请求（snmpget,snmpgetnext），或者执行多个请求（snmpwalk,snmptable,snmpdelta）。

可以用于手动设置信息的命令（snmpset）。

检索一套固定信息的命令（snmpdf，snmpnetstat，snmpstatus）。

可以把MIB oid的信息在“数字”形式和“字符”形式之间进行转换的命令（snmptranslate），它还能显示 MIB 的内容和结构。

使用 Tk/perl 来提供一个图形化的 MIB 浏览器（tkmib）。

一个接收 SNMPtrap 信息的daemon。经过选择的snmp 通知信息可以被日志记录（记录在 syslog，或者 NT 的日志，或者文本文件），转发到另一个 SNMP 管理系统，或者传递到其它的程序。

一个可扩展的代理程序（snmpd），用于对管理系统提出的 SNMP 请求做出响应。这包括了内建的多种支持性：

支持广泛的 MIB 信息模块,可以使用动态加载的模块进行扩展,可以使用外部的脚本和命令进行扩展,对多路复用 SNMP（SMUX）和代理可扩展性协议（AgentX）的支持。

*包括一个库，用于支持对新的 SNMP 开发，支持 C 和 Perl API。

Net-SNMP 对于许多的 UNIX 和类 UNIX 操作系统都是支持的，也支持 windows。注意：对于不同的系统功能会有所变化。请阅读你所在平台的 README 文件。

Net-SNMP 是什么?

包含了以下多个支持 SNMP 协议的工具：

* 一个可扩展的代理
* 一个 SNMP 库
* 用于对 SNMP 代理进行查询操作和设置操作的工具
* 用于生成和处理 SNMP traps 的工具
* 一个支持 SNMP 的‘netstat'命令
* 一个基于 Perl/Tk/SNMP 的 MIB 信息浏览器

这个包最初基于卡耐基梅隆大学的 SNMP 开发工程（version 2.1.2.1）。

9.2 Net-snmp 移植

从 Net-snmp 官方网站（http://www.net-snmp.org/）下载 Net-snmp 的源代码包。本测试是下载的比较早期的 Net-snmp 版本（net-snmp-5.0.11.2）。

下载到本地后解压。

下载：

```
wget http://jaist.dl.sourceforge.net/sourceforge/net-snmp/net-snmp-5.0.11.2.tar.gz
```

解压：

```
helight@helight:nms$ tar zxf net-snmp-5.0.11.2.tar.gz
```

配置：

```
helight@helight:nms$ cd net-snmp-5.0.11.2/ helight@helight:net-snmp-5.0.11.2$ ./configure --host=arm-linux
target=arm    --with-cc=arm-linux-gcc --with-ar=arm-linux-ar --disable-shared    --with-endianness=little
```

编译：

```
helight@helight:net-snmp-5.0.11.2$ make
```

编译后：

```
helight@helight:net-snmp-5.0.11.2$ ls -X -n apps/
total 12948
-rwxr-xr-x 1 1000 1000   664381 2008-11-06 16:29 encode_keychange
-rw-r--r-- 1 1000 1000   111137 2008-11-06 16:28 Makefile
-rwxr-xr-x 1 1000 1000   671728 2008-11-06 16:29 snmpbulkget
-rwxr-xr-x 1 1000 1000   674729 2008-11-06 16:29 snmpbulkwalk
-rwxr-xr-x 1 1000 1000   675024 2008-11-06 16:29 snmpdf
-rwxr-xr-x 1 1000 1000   670961 2008-11-06 16:29 snmpget
-rwxr-xr-x 1 1000 1000   670848 2008-11-06 16:29 snmpgetnext
drwxr-xr-x 3 1000 1000     4096 2008-11-06 16:29 snmpnetstat
-rwxr-xr-x 1 1000 1000   674177 2008-11-06 16:29 snmpset
-rwxr-xr-x 1 1000 1000   676708 2008-11-06 16:29 snmpstatus
-rwxr-xr-x 1 1000 1000   682717 2008-11-06 16:29 snmptable
-rwxr-xr-x 1 1000 1000   683799 2008-11-06 16:29 snmptest
-rwxr-xr-x 1 1000 1000   666939 2008-11-06 16:29 snmptranslate
-rwxr-xr-x 1 1000 1000   675973 2008-11-06 16:29 snmptrap
-rwxr-xr-x 1 1000 1000 1181665 2008-11-06 16:29 snmptrapd
-rwxr-xr-x 1 1000 1000   679162 2008-11-06 16:29 snmpusm
-rwxr-xr-x 1 1000 1000   681936 2008-11-06 16:29 snmpvacm
-rwxr-xr-x 1 1000 1000   674319 2008-11-06 16:29 snmpwalk
。。。
helight@helight:net-snmp-5.0.11.2$ ls -X -n agent/
total 2760
drwxr-xr-x  2 1000 1000    4096 2008-06-19 10:23 dlmods
drwxr-xr-x  3 1000 1000    4096 2008-11-06 16:29 helpers
-rw-r--r--  1 1000 1000  214531 2008-11-06 16:28 Makefile
drwxr-xr-x 21 1000 1000    4096 2008-11-06 16:29 mibgroup
-rwxr-xr-x  1 1000 1000 1635427 2008-11-06 16:29 snmpd
```

使用 file 命令查看文件类型：

```
helight@helight:net-snmp-5.0.11.2$ file agent/snmpd
```

```
agent/snmpd: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.4.3, dynamically linked
(uses shared libs), for GNU/Linux 2.4.3, not stripped
helight@helight:net-snmp-5.0.11.2$ file apps/snmpwalk
apps/snmpwalk: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux 2.4.3, dynamically linked
(uses shared libs), for GNU/Linux 2.4.3, not stripped
helight@helight:net-snmp-5.0.11.2$
```

9.3 Net-snmp 扩展

这里所扩展是说添加自己定义的管理对象，在这里也就是写一些 C 文件，然后添加到 Net-snmp 中，再进行编译，使之具有自定义的管理对象库。具体扩展的 C 文件示例可以看 agent/mibgroup/examples 这个目录中的 example.c，example.h。下面是针对于一个项目中的扩展实例。

Net-snmp 扩展后，就可以把自定义的管理对象加入到原有的系统中，这里依据最新版本 net-snmp-5.4.2.1.tar.gz 说明，其移植编译过程与上一节介绍的 net-snmp-5.0.11.2 过程相同。

下面介绍扩展步骤：

1．把编写好的代理程序（xhttp.c 和 xhttp.h）复制到目录 /usr/local/nms/net-snmp-5.4.2.1/agent/mibgroup 中。

说明：xhttp.h 较之前版本没有改动，而 xhttp.c 较之前在 ucdsnmp 中编写的程序相比，需要改动：

（1） 头文件结构；

（2） 如下图，注释部分为原有的代码，接着内容为 net-snmp 中的定义。

```
// static struct variable_list  var_trap;
static netsnmp_variable_list var_trap;

// static struct variable_list  var_obj;
static netsnmp_variable_list  var_obj;
```

详细请参看最新版本的 xhttp.c 代码。

2．在 net-snmp-5.4.2.1 目录，进行配置：./configure --host=arm-linux target=arm --with-cc=arm-linux-gcc --with-ar=arm-linux-ar --disable-shared --with-endianness=little **–with-mib-modules=”xhttp”(红色部分为扩展时必须的，这样才能加入自己的代理程序)**

3．在 net-snmp-5.4.2.1 目录，进行编译：make

4．在 net-snmp-5.4.2.1/agent 目录中，即可看到生成的 snmpd 程序，如下图：

5．把 snmpd 程序通过 NFS 复制到开发板的“/usr/bin”目录中，并用如下命令启动：

/usr/bin/snmpd –V –c /usr/snmpd.conf

说明：snmpd.conf 通过 NFS 复制到开发板的/usr 目录中，其内容如下。

```
##############################################################################
#
# snmpd.conf
#
#   - created by the snmpconf configuration program
```

```
#
##############################################################################
# SECTION: Access Control Setup
#
#     This section defines who is allowed to talk to your running
#     snmp agent.

# rocommunity: a SNMPv1/SNMPv2c read-only access community name
#     arguments:    community [default|hostname|network/bits] [oid]

rocommunity    public1

# rwcommunity: a SNMPv1/SNMPv2c read-write access community name
#     arguments:    community [default|hostname|network/bits] [oid]

rwcommunity    public2

##############################################################################
# SECTION: Trap Destinations
#
#     Here we define who the agent will send traps to.

# trapsink: A SNMPv1 trap receiver
#     arguments: host [community] [portnum]

trapsink    IP

# trap2sink: A SNMPv2c trap receiver
#     arguments: host [community] [portnum]

trap2sink    IP
```

至此，net-snmp扩展代理就实现了。

9.4 httpserver与Net-snmp接口实现

第10章 调试与编译

10.1 调试

在编写函数中，如果我们要输出调试信息，可以使用两个函数，这两个函数都定义在debug.h中，分别是：

表格 **6.1-1**

函数名	作用
debug_print()	类似于printf()函数
debug_where()	指出输出调试信息的位置(文件，行，函数名)

如果我们要添加调试信息，则在代码文件中包含debug.h头文件，当不需要的时候，则注释掉debug.h文件中的”_DEBUG_“宏定义。

10.2 编译选项

第11章 设计总结

11.1 设计总体思路

HTTP 服务器的设计思路为：

简单实用即可。对协议尽可能的简化，但也要符合标准。

系统控制部分的设计思路为：

设计为框架型，具体的功能实现以添加插件的方式来添加。

11.2 设计实现中的问题