编　　制	沈胜文
审　　核	
批　　准	
实施责任人	

Amendment history 修改历史记录

版本号	修改说明	修改批准人	修改人	日期	签收人
101	创建文档		沈胜文	2008-4-9	
102	修改文档		沈胜文	2008-4-10	

编　　制	沈胜文
审　　核	
批　　准	
实施责任人	

Table of Content 目录

1. Introduction 简介

1.1. Objective 编写目的

本文档是在《MuFTAD 网络下载需求分析》的基础上，并且结合 Wget, Prozilla, Linuxdown 程序的具体实现方式及其源代码，就下载进行的详细设计说明；

本文档将在完成需求分析文档中所要求的功能的同时，尽可能地说明下载的设计和开发，包括其开发目的和开发理论，但是设计与开发不符之处，需要讨论决定，并且修改本文档；

另外，在出错处理，协议支持（目前为 SSL）方面，不足之处，仍需要在以后的开发和使用中不断地丰富和完善。我将尽可能地对需要处理，但是目前暂不不急于开发的模块预留接口；

最终设计以代码为准；

1.2. Backgrond 背景

本程序源于法电自动下载（MuFTAD）软件项目中的一部分，负责处理 HTTP 协议请求或是下载，该软件部分以扩展库的形式组织和开发，以方便服务于其他项目；

本软件的提出者：沈胜文
本软件的开发者：沈胜文
本软件的用　户：微元科技

1.3. Terms & Abbreviation 术语&缩写

Terms&Abbreviation 术语&缩写	Description 解释

Openssl	The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.
MuFTAD	软件名称 Microunit France-Telecom Auto-Download
HTTPS	HTTPS 实际上应用了 Netscape 的安全套接字层（SSL）作为 HTTP 应用层的子层
SSL	SSL(Secure Sockets Layer)。它是支持在 Internet 上进行安全通信的标准，并且将数据密码术集成到了协议之中
HTTP	HTTP（Hyper Text Transfer Protocol），用于传输超文本标记语言文件

1.4. Reference Material 参考资料

O'Reilly.Network.Security.with.OpenSSL
An Introduction To Openssl Programming(Part I & II)
O'Reilly - HTTP Pocket Reference
openssl.org

Source Code of Wget
Source Code of Prozilla
Source Code of Linuxdown

rfc2246.pdf
rfc2617.pdf
rfc1738.pdf

2. Rules 规则

2.1. Name Rules 命名规则

HTTP(s)网络下载的实现是一个立足于 MuFTAD 项目，但是并不局限于该项目，本次开发的目的意在将 HTTP 下载开发成一个独立完整的扩展库，方便其他项目的使用；

该程序中，所有的函数均以 Mu_的形式开头，而不是 MuFTAD_。因此在整个程序内，所有的函数形名：Mu_XXX（）；

2.2. Illuminate 说明

针对每个程序，都必须注明其开发目的，开发者，开发时间，等等。以下字段必须被包含于程序的开头部分。

```
/*****************************************************************
*                        ====Microunit Techonogy Co,.LTD.====
* File Name:
*
*          XMLParse.c
*
* Description:
*
*          This file get the file name, which store the XML Contents.
*          The Functions Open the file, parse it, then return the Information we need.
*
* Revision History:
*
*          10-3-2008 ver1.0
*
*Author:
*
*          ssw    (fzqing@gmail.com)
*
*                        ***PROTECTED BY COPYRIGHT***
*****************************************************************/
```

2.3. Note Rules 注释规则

程序中的各个函数均需要明确注释其功能，并能简要描述其实现，及注意点。特别应该注意的是：在描述时，应该详细包括对锁，输入和输出进行详细说明。可参考模板

```
/*****************************************************************
*Description:
*          This Function is Parse the XML, return the Informations to caller use the
*          Value pointer;
*Input:
*          filename: the file name , which stored the XML contents
```

*Output:
*　　Pointer: which is a pointer, point to the buffer stored the Informations\
*LOCK:
*　　NONE
*Modify:
*　　ssw (fzqing@gmail.com　10-3-2008)
**********/

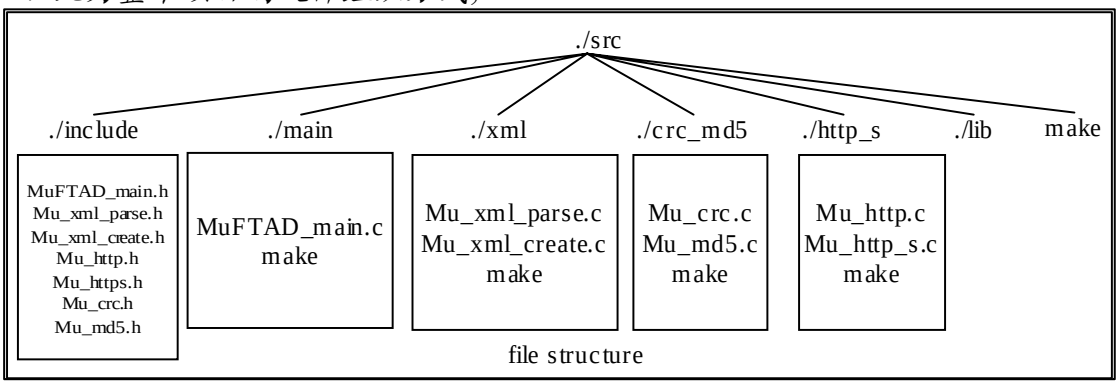
2.4. Libaries 库

待开发的扩展库必须有 openssl 库的支持，该库是一个有免费许可的用于加密、身份认证，可以轻松跨越多个平台的 C 语言库；

在大多数 Linux 操作系统上均已安装该库；

2.5. File Structure 文件组织

[注意]:
1、此为整个项目的文件组织方式;



./src/include：文件夹，包含该项目中的所有头文件；
./src/main：文件夹，包含所有按法电《do3c00_SoftProtocol_0.1.0_RC1》流程所开发的程序；
./src/xml：文件夹，包含项目中所需要的 xml 处理库函数源代码；
./src/crc_md5：文件夹，包含项目中所需要的校验函数源代码，包括 CRC 和 MD5 校验代码；
./src/http_s：文件夹，包含项目中所需要的与服务器交互的方式，包括 HTTP(s) GET、POST 方式；
./src/lib：文件夹，用于存储编译所生成的 xml、http 和 https、crc/md5库。软件编译连接时使用该文件夹下的库；
./src/make：文件，总的编译入口；
[注意]:

1、各对应文件夹下的源文件按需要添加，但是所作修改必须对 makefile 文件作相应的修改，以正确编译;

3. Structure Of Routines 程序结构

3.1. Overview 总述

该程序需要处理的二种情况包括：

1、HTTP(s)查询

该部分提供一种查询功能，以便于 Device 查询服务器上的一些信息，因为 Device 在与服务器交互时，所使用的信令内容一般比较小，整个交互仅仅使用单线程完成；

同时，信令的交互也不提供续传，记录等功能，当查询失败后，只能放弃，无法进行其他重试操作；

在本项目中，信令的交互基本上全部需要 Openssl 的支持；

2、HTTP(s)下载

该部分用于从服务器端下载节目或其他内容，该部分下载的内容通常不同于信令，内容较大，通常也不需要加密；但是在本项目中，需要提供其他额外的，更好的方式来下载 HTTP；

需要提供的功能有：续传、记录、对于 HTTPS 类型，本项目暂不考虑实现，但是必须预留线程函数接口，在后期再完善；

3.2. Routines List 函数列表

4. Global Description 全局描述

4.1. Global Type 全局类型

4.1.1. Macro 宏定义

#define MAX_THD 10

定义最大线程数

#define MAX_THD_LEN 0x100000
定义最小文件块大小

#define CHECK_STATOK(x) ((x) >= 200) && ((x) < = 299)
检查服务器返回的状态值，若请求成功，该宏返回真

#define CHECK_STATRE(x) ((x) == 303)
若服务器返回状态值表明重定向，该宏返回真

4.1.2. Structure 结构体

```
enum _Mu_Urlscheme{
    URL_NONE,
    URL_HTTP ,
    URL_HTTPS
}Mu_Urlscheme
```
定义协议的类型；

```
    typedef struct _Mu_URL{
        char *url;
        char *host;
        char *path;
        char *file;
        char *username;
        char *password;
        enum url_scheme scheme;
        int port;

        int urllen;
        int hostlen;
        int pathlen;
        int filelen;
        int usernamelen;
        int passwordlen;
    }Mu_URL;
    Mu_URL mu_url;
```
定义一个结构，以存储 URL 信息。在本程序取得 URL 地址后，需要解析出该地址中所有信息；

url：URL 完整地址，长度为 urllen；
scheme：标记是 HTTP 还是 HTTPS 下载；
host：标记主机；
port：标记端口；
path：标识请求地址路径；

file：标识请求文件名称；
username：请求服务器时用的用户名；
password：请求服务器时用的密码；
[注意]:
1、对于该结构内的指针，均在程序初始化时被设置，初始时，分配一个默认大小，在后期存储相应内容时，若不够，再扩大相应的存储空间;
2、对于无内容字段，相应的内容空间全部置为 NULL;
3、程序只维持一个该结构，同一时刻只解析出一个 URL 并发起连接，或采用单线程，或采用多线程;

```
typedef sturct _Mu_Httpstatus{
    thread_t threadnumber;
    off_t seekstart;
    off_t seeknew;
    off_t bytesleft;
    off_t bytesexpect;
    int retry;

    int fd;
    int statcode;
    int threadstatuse;
    struct timeval starttime;
    struct timeval dltime;
    int rates;
}Mu_HttpStatus, *Mu_HttpStatusPtr;
```

说明：
threadnumber：线程号，一个线程对应一个线程号，但是并不是线程 ID，该线程号对应于 Mu_HttpStatus 数组中的下标号；
seekstart：用于定位该线程写入文件的开始位置；
seeknew：当写入文件后，应该更新该值，用于继续下载；
bytesleft：表明该线程还有多少内容没有下载；
bytesexpect：线程原分配的有多少内容需要下载；
retry：重试次数；
fd：待写入的文件句柄；
statcode：该线程运行时 HTTP 返回的状态码；
threadstatuse：线程运行状态，表明线程是处于运行还是中止状态；
starttime：该线程运行开始时间；
dltime：线程写数据时的时间；
rates：下载速率，（bytesexpect – bytesleft）/(dltime - starttime)

```
char *location;
```

location：线程运行过程中，用于存储重定位地址。重新存储地址时，需要先释放该空间；

用该空间重定向地址重新下载后，需要清空该空间值，并将该值设置为 NULL；

```
typedef enum _Mu_DnType{
    MU_QUERY = 1,
    MU_DOWNLOAD = 2
}
```

用于标识 HTTP 请求服务器时有类型，将下载和信令查询区分开；

```
typeded struct _Mu_DnStat{
        enum Mu_BuildMethod type;
        int fd;
        int xmlfd;
        off_t startops;
        off_t length;
        int resume;
    }Mu_DnStat,   *Mu_DnStatPtr;
```

查询服务器或是下载前，调用者除了需要提供 URL 地址信息外，还必须提供上述结构内的信息；

type：查询类型；

fd：待写入的文件句柄；

xmlfd：POST 方式提交信令时，信令存储文件；

startops：待写入文件 fd 的起始地址；

length：总体长度信息；

resume：是否续传；

```
typedef enum _Mu_BuildMethod{
    MU_GET = 1,
    MU_HEAD,
    MU_POST
}Mu_BuildMethod;

typedef struct _Mu_Build{
    Mu_BuildMethod method;
    int fd;
    off_t startops;
    off_t endops;
}Mu_Build
```

用于构建请求报文

method：请求报文的格式；

fd：用于构建 POST 请求报文时的信息；

startops：请求报文开始长度；
endops：请求报文的结束长度；
[注意]:
1、 若不是 POST，fd 可以为 NULL;
2、若不用分线程处理，endops 应该为0;

```
typedef enum _Mu_DnStatCode{
    MU_DNCNT = 1,
    MU_DNRCV,
    MU_DNCMP,
    MU_RECNT,
}
```

用以标明线程在下载或是查询服务器时，从服务器端获得报文的状态：
MU_DNCNT：连接服务器阶段；
MU_DNRCV： 接收数据包阶段；
MU_DNCMP： 接收完成；
MU_RECNT： 重新连接服务器阶段；

4.2. Global Error 全局错误码

[注意]:
1、该错误码应该被放在一个单独的，只用来定义错误的头文件中;
2、该头文件定义为：mu_error.h

```
#define MUOK 0
#define MUNBUF -1
#define MUNHST -2
#define MUNPAT -3
#define MUNLEN -4
#define MUNLVR -5
#define MUNCOK -6
#define MUNPOS -7
#define MUNRLC -8
#define MUNCMP -9
#define MUNSPA -10

#define MUEAUH -11
#define MUESND -12
#define MUERCD -13
#define MUEPRO -14
#define MUERED -15
#define MUEEOF -16
```

```
#define MUECON -17
#define MUEQUE -18
#define MUEDEL -19
#define MUEURL -20
#define MUEFSY -21
#define MUEMEH -22
#define MUERLC -23
#define MUESSL -24
……
```

5. Routines Details 函数细节

5.1. Mu_CheckProtoc

5.1.1. Name 函数名称

int Mu_CheckProtoc(const char *url)

5.1.2. Description 函数描述

解析 URL 内容时，首先需要解析其协议类型，对于不支持的协议类型，函数以出错返回；

对于解析出来的协议，修改全局值 Mu_URL 结构中的 type 字段值，并同时修改 MuIO 中的 type 字段，以便于在调用 IO 函数时，选择合适的操作函数；

目前，只支持二种协议类型：**HTTP 和 HTTPS**；
对于其他协议，均以出错返回；

5.1.3. Function 功能

解析 URL 中的协议字段，获得相应的协议类型；

5.1.4. Capability 性能

略

5.1.5. Input 输入

url：调用者传入的一个 URL 字串地址，该字串应该包括完整的请求地址；该字串所使用的内存空间的释放由调用者完成；

5.1.6. Output 输出

返回处理的状态码，包括：
1、解析正确，以 **MUOK** 返回；
2、不支持的协议，以 **MUEPRO** 返回；

5.1.7. Arithmetic 算法

完整的 URL 如下示：
http(s)://username:password@hostname:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
url: http(s)://username:password@hostname:port/path1/path2/filename
scheme：MU_HTTP 或是 MU_HTTPS；

5.1.8. Process 处理流程

略

5.1.9.Pseudocode 伪代码

```
char *check_protoc[] = {
        “http:”,
        “https:”,
        NULL
      }

int Mu_CheckProtoc(const char *url)
{
      char **s;
      int i = 0;

      for(s = check_protoc; *s; s++, i++){
```

```
        if(strncasecmp(url, *s, strlen(*s)) == 0){
            if(i){
                mu_url.scheme = URL_HTTPS;
                MuIO.type = MU_HTTPS;
            }else if(!i){
                mu_url.scheme = URL_HTTP;
                MuIO.type = MU_HTTP;
            }else
                return MUNPRO;

            return MUOK;
        }

    return MUEPRO;
}
```

5.2. Mu_CheckUser

5.2.1. Name 函数名称

int Mu_CheckUser(const char *url)

5.2.2. Description 函数描述

该函数从 URL 地址中取用户名和密码，若连接使用 Auth 方式时，取相应值放入内存段；

若 URL 中不存在相应的用户名和密码，那么 Mu_URL 结构中的 username 和 password 字段需要被清空；

用户名和密码将会采用 **base64加密**后构建 HTTP 请求报文头；

5.2.3. Function 功能

从 URL 地址中解析出用户名和密码

5.2.4. Capability 性能

略

5.2.5. Input 输入

同5.1.5.中说明

5.2.6. Output 输出

返回处理的状态码，包括：
1、解析正确，以 **MUOK** 返回；
包括二种情况：存在 username 和不存在 username;
存在时，username 和 password 字段长度非空；否则该指针指向地址内容为空；

2、存储出错，以 **MUNBUF** 返回；

3、格式错误，以 **MUEPRO** 返回；

5.2.7. Arithmetic 算法

完整的 URL 如下示：

第一种情况：
http(s)://username:password@hostname:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
username：username
password：password
以//开始，以第一个@结束，且其中有（:）的字段即为用户密码字段，以 **MUOK** 返回；

第二种情况：
http(s)://username@hostname:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
username：username
password：NULL
以//开始，以第一个@结束，无密码字段，以 **MUOK** 返回；

第三种情况：
http(s):// hostname:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
username：NULL
password：NULL
　　无用户名和密码字段，以 **MUOK** 返回；

第四种情况：
http(s)://:password@hostname:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
username：NULL
password：password
　　无用户名，但有密码字段，以协议错 **MUEPRO** 返回；

5.2.8. Process 处理流程

略

5.2.9. Pseudocode 伪代码

```
Mu_URL mu_url;
int Mu_CheckUser(const char *url)
{
    char *p = NULL;
    char *q = NULL;
    char *s = NULL;

    //clear the buffer
    //if no username and password, make sure the length of buffer is ZERO;
    memset(mu_url.username, 0, mu_url.usernamelen);
    memset(mu_url.password, 0, mu_url.password);

    //judge whether contain the username & password
    if((NULL != (p = strstr(url, "//"))){
        P += 2;

        //if have no username and password,
        //return MUOK, but the buffers are empty
        if(NULL == (q = strchr(p, '@'))){
            fprintf(stdout, "there are not username & password!\n");
            return MUOK;
        }
```

```
//there are username and passwrod
if((NULL != (s = strchr(p, ':')))
    &&(s < q)&&( p != s){

    //stored the username
    if(s > (p+1)){
        if((s-p) >= mu_url.usernamelen){
            if(NULL == realloc(mu_url.username, (s-p+1)))
                return MUNBUF;
            mu_url.usernamelen = s-p+1;
            memset(mu_url.username, 0, mu_url.usernamelen);
        }
            snprintf(mu_url.username, s-p, "%s",    p);
    }

    //store the passwrod
    if(q > (s + 1)){
        if((q-s-1) >= mu_url.passwordlen){
            if(NULL == realloc(mu_url.password, (q-s)))
                return MUNBUF;
            mu_url.passwordlen = q-s;
            memset(mu_url.password, 0, mu_url.password);
        }
            snprintf(mu_url.password, q-s-1, "%s",    s+1);
    }

    return MUOK;
}else if(s == NULL){

//there are username, but not password
    if(q > (p + 1)){
        if((q-p) >= mu_url.usernamelen){
            if(NULL == realloc(mu_url.username, (q-p+1)))
                return MUNBUF;
            mu_url.usernamelen = q-p+1;
            memset(mu_url.username, 0, mu_url.usernamelen);
        }
        snprintf(mu_url.username, q-p, "%s",    p);
    }
    return MUOK;
}else
    return MUEPRO;

}else
```

```
        return MUEPRO;
}
```

5.2.10. Interface 接口

略

5.2.11. Malloc 存储分配

结构 mu_url 所维护的成员中，包括指针和长度值，二个一组，用以表示各指针所指向内存空间的内存；

该结构中成员所指向的地址空间不会被释放，也不会减小，但是会根据成员大小而**增加内存空间**；

5.2.12. Restrict 限制

略

5.2.13. Test 测试

略

5.2.14. Unsolve 未解决情况

略

5.3. Mu_GetHost

5.3.1. Name 函数名称

int Mu_GetHostPort(const char *url)

5.3.2. Description 函数名称

该函数从 URL 中取出主机名，不管是主机名，还是 IP 地址，均需要能正确地从 URL 中取出地址；

取出的地址存放于 Mu_URL 结构中的 host，host 的大小由 hostlen 标识；

5.3.3. Function 功能

从 URL 中取出主机名

5.3.4. Capability 性能

略

5.3.5. Input 输入

同5.2.5.中说明

5.3.6. Output 输出

返回函数的处理状态
1、解析正确，以 **MUOK** 返回；
2、无主机地址，以 **MUNHST** 返回；
3、存储出错，以 **MUNBUF** 返回；
4、协议错，以 **MUEPRO** 返回；

5.3.7. Arithmetic 算法

完整的 URL 如下示：
第一种情况：
http(s)://username:password@hostname:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
host：hostname
port：port

介于//与第一个/之间；
1、若其中存在@，存在用户名和密码，跳过该部分；
2、若其后有：，则将值存入 port；

第二种情况：
http(s)://username:password@hostname/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
host：hostname
port：port
介于//与第一个/之间；
1、若其中存在@，可能存在用户名和密码，跳过该部分；
2、无（：），port 置为0；

第三种情况：
http(s)://username:password@:port/path/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
host：NULL
port：port
介于//与第一个/之间；
1、若其中存在@，存在用户名和密码，跳过该部分；
2、 ：前无内容，host 为 NULL；其后有：，则将值存入 port；
该情况以 **MUEPRO** 返回；

5.3.8. Process 处理流程

略

5.3.9. Pseudocode 伪代码

```
int Mu_GetHostPort(const char *url)
{
    char *p = NULL;
    char *q = NULL;
    char *s = NULL;
    char count[6];     //use to store port
    int i = 0;

    memset(count, 0, 6);
    memset(mu_url.host, 0, mu_url.hostlen);
```

```
mu_url.port = 0;

if(NULL != (p = strstr(url, "//"))){

    //have the host segments
    if((P += 2) &&(NULL != (q = strchr(p, '/')) && ((q – p) != 1)){

        //skip the username and password
        if((NULL != (s = strchr(p, '@')))
            && (s < q-1)){
            fprintf(stdout, "skip the username&password to got hostname\n");
            p = s + 1;
        }else if(NULL == s)
            fprintf(stdout, "have no username and password\n");
        else if(s >= q){
            fprintf(stdout, "usename have the char /, U should omit it!\n");
            return MUEPRO;
        }

        //get ride of the port & hostname
        if((NULL != (s = strchr(p, ':')))
            &&(s < q)){
            if( s > (p+1)){
                if((s-p) >= mu_url.hostlen){
                    if(NULL == realloc(mu_url.host, s-p+1))
                        return MUNBUF;

                    //bzero
                    mu_url.hostlen = s-p+1;
                }
                snprintf(mu_url.host, s-p, "%s", p);
            }
            //anaylise the port
            if((q > (s + 1)){
                if((q-s-1) <= 6){
                    snprintf(count, q-s-1, "%s", s);
                    mu_url.port = atoi(count);
                }
            }
        }else if(s == NULL){
            if(q > ( p + 1)){
                if((q-p) >= mu_url.hostlen){
                    if(NULL == realloc(mu_url.host, q-p+1))
                        return MUNBUF;
                    //bezero
```

```
                    mu_url.hostlen = q-p+1;
                }
            snprintf(mu_url.host, q-p, "%s", p);
            }
        }else
            return MUEPRO;

        return MUOK;

    }else
        return MUNHST;
}else
    return MUEPRO;

}
```

5.3.10. Interface 接口

该解析函数只接收一个 URL 地址，解析其中的各项信息，存储于全局结构 mu_url 中，用该结构中的成员 host 指针指向存储主机地址的空间，以 hostlen 标记**可用内存**大小；

[注意]:

1、可用内存大小，是指用来存储主机地址的内存区域，而不是指实际存储的内容长度;

mu_url 结构中的 host 指针，若解析 URL 时，不能获得 host 内容，指针指向区域的内容为0，这也是为什么解析前必须置0的原因；

5.3.11. Malloc 存储分配

同5.2.11.中说明

5.3.12. Restrict 限制

略

5.3.13. Test 测试

略

5.3.14. Unsolve 未解决情况

该解析函数只解决了正规的 URL 地址，即**必须存在//和/**，若不存在//和/，解析地址会以 **MUEPRO** 协议错返回；

解析函数决定了主机段不可用/和@字符，否则解析出内容不可用；

5.4. Mu_GetPathFile

5.4.1. Name 函数名称

int Mu_GetPathFile(const char *url)

5.4.2. Description 函数描述

从 URL 路径中取得访问的内容在服务器上的路径，包括文件名；**路径名**存入 Mu_URL 结构中的 path 成员中，**文件名**则存入结构中的 file 成员中；
Path 的大小由 pathlen 标识，file 的大小由 filelen 标识；

5.4.3. Function 功能

从 URL 地址中获取待访问文件在服务器上的路径及文件名

5.4.4. Capability 性能

略

5.4.5. Input 输入

同5.2.5.中说明

5.4.6. Output 输出

返回函数的处理状态
1、解析正确，以 **MUOK** 返回；

2、存储出错，以 **MUNBUF** 返回；
3、无文件名，以 **MUNFIL** 返回；
4、协议错，以 **MUEPRO** 返回；

5.4.7. Arithmetic 算法

完整的 URL 如下示：
第一种情况：
http(s)://username:password@hostname:port/path1/path2/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
path：path1/path2
　　介于第一个/和最后一个/之间的内容；
　　[注意]:
　　1、解析时在获得第一个/前，应跳过//，否则解析出错;
filename：filename
　　最后一个/后的内容

第二种情况：
http(s)://username:password@hostname:port/filename

解析完成后，Mu_URL 结构中的相应成员值如下示：
path：NULL；
filename：filename
　　最后一个/后的内容

第三种情况：
http(s)://username:password@hostname:port
path：NULL；
filename：NULL
　　[注意]:
　　*1、这种情况现以**出错返回**;*

5.4.8. Process 处理流程

略

5.4.9.Pseudocode 伪代码

```
int Mu_GetPathFile(const char *url)
{
    char *p = NULL;
    char *q = NULL;
    char *s = NULL;

    //clear the buffer of path&file
    memset(mu_url.path, 0, mu_url.pathlen);
    memset(mu_url.file, 0, mu_url.filelen);

    //skip the "//"
    if((NULL == (s = strstr(url, "//"))) || !(*(s += 2)))
        return MUEPRO;

    //get path
    if((NULL != (p = strchr(s, '/'))) && (NULL != (q = strrchr(s, '/')))){

        //have path
        //if there is NOT path, we should skip and DO NOT return error!
        if(q > (p + 1)){
            if((q-p-1) >= mu_url.pathlen){
                if(NULL == realloc(mu_url.path, q-p));
                    return MUNBUF;
                //bezero
                mu_url.pathlen = q-p;
            }
            snprintf(mu_url.path, q-p-1, "%s", p+1);
        }

        //get filename
        if((q+1) != NULL){
            if(strlen(q+1) >= mu_url.filelen){
                if(NULL == realloc(mu_url.file, (strlen(q+1)+1));
                    return MUNBUF;
                //bezero
                mu_url.filelen = strlen(q+1)+1;
            }
            snprintf(mu_url.file, strlen(q+1), "%s", q+1);
        }else
            return MUNFIL;

        return MUOK;
```

```
    }else
        return MUEPRO;
}
```

5.4.10. Interface 接口

该函数处理路径和文件名，在本函数实现时，对路径是可有可无的，但是**必须有文件名的存在**，要不然，会以出错返回；

解析前，mu_url 结构中的 path 和 file 指针所指向的内容都必须被清空，以便于在合成请求地址时正确使用；

上述二个地址空间不会缩小，但是会随着需要而增大，即空间大小始终是所有解析的字段的最大大小；

5.4.11. Malloc 存储分配

同5.2.11.中说明

5.4.12. Restrict 限制

略

5.4.13. Test 测试

略

5.4.14. Unsolve 未解决情况

对于URL中地址的解析，并未包含所有的情况，例如：http://username:passwrod@hostname:port/name/这样的地址，将会被解析出错，以MUEPRO返回，表明协议地址错；

本函数可解析的格式均为本项目需要，并且最多如此的情况；

5.5. Mu_ParseUrl

5.5.1. Name 函数名称

int Mu_ParseUrl(const char *url)

5.5.2. Description 函数描述

程序进行处理的第一步，便是解析从调用者处获得的 URL 地址，从中解析出相应的信息，以进行更进一步的处理；

根据 URL 字串内容，解析出的信息存储于 Mu_URL 结构中，该结构中的所有字段在初始化时均被分配一定的内存区域，若解析出的内容超出，则扩充该区域，但是并不会减少内存区域；

整个程序只维持一个该变量，即程序同时只处理一个 URL 连接，即使采用多线程连接服务器；

对于 URL 中不包括的信息，如 username，password，path 需要清空（全部置 0）；

5.5.3. Function 功能

解析 URL 字串，填充入 Mu_URL 结构中，以方便进一步的处理；

5.5.4. Capability 性能

略

5.5.5. Input 输入

同5.1.5.中说明

5.5.6. Output 输出

返回处理的状态码，包括：

1、解析正确，以 **MUOK** 返回；
2、解析错误时，返回上述解析模块的错误代码；

5.5.7. Arithmetic 算法

完整的 URL 如下示：
http(s)://username:password@hostname:port/path/filename
解析完成后，Mu_URL 结构中的相应成员值如下示：
url: http(s)://username:password@hostname:port/path1/path2/filename
username：username
password：password
以//开始，以第一个@结束，且其中有：的字段即为用户密码字段；否则不存在用户名和密码；

host：hostname
port：port
介于//与第一个/之间；
若其中存在@，存在用户名和密码，跳过该部分；
若其后有：，则将值存入 port；

path：path1/path2
主机名之后的/起，最后一个/前的部分均为 path；
处理时应该检测二个/的位置是否相同，若相同，则放弃；
file：filename
最后一个/开始至结尾；
[注意]:
1、解析过程中的的错误处理，以各实现函数为准;

5.5.8. Process 处理流程

略

5.5.9. Pseudocode 伪代码

```
int Mu_ParseUrl(const char *url)
{
    int ret = MUOK;

    if(((ret = Mu_CheckProtoc(url)) < 0)
```

```
        ||((ret = Mu_CheckUser(url) ) < 0)
        ||((ret = Mu_GetHostPort(uri)) < 0)
        ||((ret = Mu_GetPathFile(url)) < 0))

        fprintf(stdout, “parse the file error!!!”);

    return ret;
}
```

5.5.10. Interface 接口

该函数的功能即调用其他函数来解析 URL 地址，并处理各种错误码，使用的函数有：

Mu_CheckProtoc;
Mu_CheckUser;
Mu_GetHostPort;
Mu_GetPathFile;

5.5.11. Malloc 存储分配

解析时需要的内存空间，由各个解析函数实现，各个内存空间由 mu_url 结构中的指针指向；

5.5.12. Restrict 限制

略

5.5.13. Test 测试

略

5.5.14. Unsolve 未解决情况

略

5.6. Mu_BuildQuery

5.6.1. Name 函数名称

int Mu_BuildQuery(Mu_Build build, char **Ptr)

5.6.2. Description 函数描述

该函数主要用于创建请求报文，在本项目中，需要的请求报文方式有三种：**GET、HEAD 和 POST**；

GET：用于下载和普通请求；

HEAD：用于在多线程下载时，不知道节目长度情况下，获取节目内容长度；

POST：请求服务器时，提交信令；

本函数本身不创建请求报文，仅仅是由多个创建函数组成，向外提供一个统一接口；

5.6.3. Function 功能

创建请求报文；

5.6.4. Capability 性能

略

5.6.5. Input 输入

build：Mu_Build 结构体，成员参考4.1.2.中说明

Ptr：由函数分配的内存空间，采用二级指针作为参数返回空间；

[注意]:

1、Ptr 的释放由调用者完成;

5.6.6. Output 输出

函数处理状态码，包括：

1、与调用的子函数返回码相同；
2、不包含的请求方式，将以 **MUEBUL** 返回；

5.6.7. Arithmetic 算法

略

5.6.8. Process 处理流程

略

5.6.9. Pseudocode 伪代码

```
Mu_Build build;
int Mu_BuildQuery(Mu_build build, char **Ptr)
{
    int ret = MUOK;

    switch(build.method){
        case MU_GET:
            ret = Mu_BuildQueryGet(build, Ptr);
            break;
        case MU_HEAD:
            ret = Mu_BuildQueryHead(build, Ptr);
            break;
        case MU_POST:
            ret = Mu_BuildQueryPost(build, Ptr);
            break;
        default:
            return MUEBUL;
    }

    return ret;
}
```

5.6.10. Interface 接口

函数在完成功能时，利用的接口有：
Mu_BuildQueryGet;
Mu_BuildQueryHead;

Mu_BuildQueryPost;

在调用本函数创建请求报文的时候，需要提供参数有：Mu_Build 结构体和一个二级指针；
结构体中的成员名称及相关意义在4.1.2.中说明，调用者必须按需要填充各值；
二级指针实际上是作为一个分配存储空间的一个返回值；

本函数的返回值，在出错时均为负值；

5.6.11. Malloc 存储分配

略

5.6.12. Restrict 限制

略

5.6.13. Test 测试

略

5.6.14. Unsolve 未解决情况

本函数现阶段只处理 HTTP 的 **GET，POST，HEAD** 三种报文，对于其他类型的报文，程序现暂不支持；

对于不支持的报文格式，函数将以 **MUEBUL** 返回；

若要实现对其他报文格式的支持，必须满足下列三个要求：
1、在 Mu_BuildMethod 枚举类型中添加相应类型值；
2、在本函数中设置相应的跳转；
3、用于创建该报文的函数，类似于 Mu_BuildQueryGet；

5.7. Mu_BuildQueryGet

5.7.1. Name 函数名称

int Mu_BuildQueryGet(Mu_Build build, char **Ptr)

5.7.2. Description 函数描述

创建 GET 请求报文，并以 Ptr（二级指针）返回存储有请求报文的内存空间;

GET 方式主要用于分线程下载文件，因此需要提供 **Range** 字段来构建 HTTP 请求报文；

5.7.3. Function 功能

创建 GET 请求报文

5.7.4. Capability 性能

略

5.7.5. Input 输入

同5.6.5. 中说明

5.7.6.Output 输出

在创建报文时，返回函数运行过程中的状态，包括：
1、无错误，以 **MUOK** 返回；
2、分配内存空间失败，以 **MUNBUF** 返回；
3、校验出错，以 **MUEAUH** 返回；

5.7.7. Arithmetic 算法

在创建请求报文时，必须明确报文中必须包含的内容字段，现阶段，必须包括以下内容：

1、User-Agent：指明 Device 的访问工具；

2、Host：主机字段；

3、Accept：表明 Device 可以接受的内容类型，在本项目中，该字段设置为*/*，以表明可接受所有类型；

4、Cookie：若全局 cookies 指针非空，值为 cookies 所指内容；否则为空；

5、Authorization：用于简单的用户认证，用于形如：http://username:password@host类型的请求；

若 mu_url 结构中 username 和 password 字段均为空，Authorization 字段也为空，否则是运算后的 **Base64**值；

6、Rage：标识出 Device 欲请求的内容，用于多线程下载，若不需要多线程下载，该字段应该被忽略；

7、Connection：表明连接状态，在本程序中，设置为 Keep-Alive；

5.7.8. Process 处理流程

略

5.7.9. Pseudocode 伪代码

```
char *cookies = NULL;
char *Mu_GetBasicAuth(const char *username, const char *password);
#define USER_AGENT “Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)”

int Mu_BuildQueryGet(Mu_Build build, char **Ptr)
{
    int ret = MUOK;
    int reqlen;
    char *authuser = NULL;

    //get the auth
    if(strlen(mu_url.username) || strlen(mu_url.password)){
        if(NULL == (authuser =
                        Mu_GetBasicAuth(mu_url.username, mu_url.password)))
            return    MUEAUH;
    }
```

```
//define the format of the request packet
const char reqfmt[]=
“GET %s/%s HTTP/1.0\r\n”
“User-Agent: %s\r\n”
“Host: %s\r\n”
“Accept: */*\r\n”
“Cookie: %s\r\n”
“Authorization: BASIC %s\r\n ”
“Range: bytes=%Ld-%s \r\n”
“Connection: Keep-Alive\r\n”
“\r\n”;

//get the length of the query buffer
reqlen = snprintf(NULL, 0,
        reqfmt,
        mu_url.path, mu_url.file,
        USER_AGENT, mu_url.host,
        Cookies? Cookies: “”,
        authuser? authuser: “”,
        build.startops,
        (build.endops && ltoa(build.endops))? ltoa(build.endops): “”);

if(reqlen < = 0){
    do error;
    goto error;
}

reqlen ++;
if(NULL == ((char *)*Ptr = (char *)malloc(reqlen))){
    do error;
    goto error;
}

memset(*Ptr, 0, reqlen);
snprintf(*Ptr, reqlen - 1,
        reqfmt,
        mu_url.path, mu_url.file,
        USER_AGENT, mu_url.host,
        Cookies? Cookiew: “”,
        authuser? authuser: “”,
        build.startops,
        (build.endops && ltoa(build.endops))? ltoa(build.endops): “”);

if(authuser)
    free(authuser);
```

```
    return MUOK;

error:
    if(authuser)
        free(authuser);

    return MUNBUF;
}
```

5.7.10. Interface 接口

在完成上述功能时，需要用到一个函数接口用于完成 base64加密：
Mu_GetBasicAuth；

另参考5.6.10.中说明

5.7.11. Malloc 存储分配

构建请求报文时，所使用的空间是在函数中动态分配的，在申请空间前，先检测出所需要使用的空间，然后再申请空间以存储报文内容；

该函数动态分配的内存空间需要被调用者在使用结束后释放；

若需要认证，则必须用 Mu_GetBasicAuth 函数来计算 username 和 password 的 Base64加密值，该值存放于由该函数申请的内存空间，但是该空间的释放由 Mu_BuildQueryGet 完成；

5.7.12. Restrict 限制

略

5.7.13. Test 测试

略

5.7.14. Unsolve 未解决情况

现阶段的报文头不提供 reference，Accept-Language，Accept-Encoding，Content-Length 等内容；

5.8. Mu_BuildQueryHead

同5.7.中说明的 Mu_BuildQueryGet，但是也存在不同之处，本函数中，不需要设置 Range 字段；

该函数主要用于对一个节目进行分线程处理时，先取得节目的整体大小；
函数名称为：
int Mu_BuildQueryHead(Mu_Build build, char **Ptr);

5.9. Mu_BuildQueryPost

5.9.1. Name 函数名称

int Mu_BuildQueryPost(Mu_Build build, char **Ptr)

5.9.2. Description 函数名称

构建 POST 请求报文，在本项目中，需要用些方式构建包括 XML 信令的请求报文以此与服务器交互；

构建 POST 请求报文头时，需要从 build 中获得文件句柄把描述的文件的长度，以此来填充报文头中的 Content-Length 字段；

5.9.3. Function

构建 POST 请求报文

5.9.4. Capability 性能

略

5.9.5. Input 输入

同5.7.5.中说明

5.9.6. Output 输出

同5.7.6.中说明，但是增加一个错误：
MUNPOS：当无 POST 信令无件时，返回该值；

5.9.7. Arithmetic 算法

POST 报文头的构建，不包括正文段部分，在 POST 请求服务器时，需要将 fd 内的内容额外发送给服务器；

报文头的构建类似于 Mu_BuildQueryGet 函数，但是多了一个 Content-Length 字段，以标识 POST 的正文段的内容长度；

POST 在发送完正文段后，需要额外再发送二个字符以结束该部分（\r\n）；
[注意]:
1、报文头中的 Content-Length 不包括尾部的\r\n;
2、POST 报文段格式应该如下:
报文头
\r\n
\r\n
正文段
\r\n

5.9.8. Process 处理流程

略

5.9.9. Pseudocode 伪代码

```
char *cookies = NULL;
char *Mu_GetBasicAuth(const char *username, const char *password);
#define USER_AGENT “Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)”
```

```
int Mu_BuileQueryPost(Mu_Build build, char **Ptr)
{
    int ret = MUOK;
    int reqlen;
    char *authuser = NULL;
    struct stat info;

    //get the auth
    if(strlen(mu_url.username) || strlen(mu_url.password)){
        if(NULL == (authuser =
                        Mu_GetBasicAuth(mu_url.username, mu_url.password)))
            return MUEAUH;
    }

    if((build.fd == -1) || (fstat(fileno(build.fd), &info) == -1))
        return MUNPOS;

    const char reqfmt[]=
    "POST %s/%s HTTP/1.0\r\n"
    "User-Agent: %s\r\n"
    "Host: %s\r\n"
    "Accept: */*\r\n"
    "Cookie: %s\r\n"
    "Content-Length: %Ld\r\n"
    "Authorization: BASIC %s\r\n "
    "Connection: Keep-Alive\r\n"
    "\r\n\r\n";

    reqlen = snprintf(NULL, 0,
            reqfmt,
            mu_url.path, mu_url.file,
            USER_AGENT, mu_url.host,
            Cookies? Cookies: "",
            info.st_size,
            authuser? authuser: "");

    if(reqlen < = 0){
        do error;
        goto error;
    }

    reqlen ++;
    if(NULL == ((char *)*Ptr = (char *)malloc(reqlen))){
        do error;
        goto error;
```

```
    }

    memset(*Ptr, 0, reqlen);
    snprintf(*Ptr, reqlen - 1,
            reqfmt,
            mu_url.path, mu_url.file,
            USER_AGENT, mu_url.host,
            Cookies? Cookiew: “”,
            info.st_size,
            authuser? authuser: “”);

    if(authuser)
        free(authuser);

    return MUOK;

error:
    if(authuser)
        free(authuser);

    retrun MUNBUF;

}
```

5.9.10. Interface 接口

在完成上述功能时，需要用到一个函数接口用于完成 base64加密：
Mu_GetBasicAuth;

另参考5.6.10.中说明

5.9.11. Malloc 存储分配

同5.7.11.中说明；

5.9.12. Restrict 限制

略

5.9.13. Test 测试

略

5.9.14. Unsolve 未解决情况

略

5.10. Mu_FetchHeader

5.10.1. Name 函数名称

int Mu_FetchHeader(int socket, char **hdr)

5.10.2. Description 函数描述

该函数从套接口 socket 中读取报文，存放于 hdr 所指向的内存空间中；
[注意]：
1、一次只读取一行；

5.10.3. Function 功能

读取报文

5.10.4. Capability 性能

略

5.10.5. Input 输入

socket：套接字，用于读取报文头；
hdr：二级指针，用于返回本函数所申请的，用于存放报文的内存空间；

5.10.6. Output 输出

函数运行状态，包括
1、无错误，以 **MUOK** 返回；
2、申请内存空间出错，以 **MUNBUF** 返回；
3、读报头出错，以 **MUERED** 返回；
4、读到报文的的尾部，即后面不再有数据，以 **MUEEOF** 返回；

5.10.7. Arithmetic 算法

在读取 Socket 内的内容时，为了便于解析服务器返回的报文头，采取了逐行接收，逐行解析的方式；

对于服务器返回的报文头，每行后都对应一个“\r\n”；
[注意]:
1、在读取一行时（准确说是一个字段），应该考虑到服务器为了工整，将一个比较长的字段分成若干行写入，但是前提是后续行以\t 制表位或是空格开头;
例如:

```
HTTP/1.0 200 OK
[Server headers here]
Set-Cookie: acct=04382374;
        domain=.ora.com;
        Expires=Sun, 16-Feb-2003 04:38:14 GMT;Path=/
```

5.10.8. Process 处理流程

略

5.10.9. Pseudocode 伪代码

```
#define MU_MAX_HEADER 1024
int Mu_FetchHeader(int socket, char **hdr)
{
    int i, bufsize;
    int ret = MUOK;
    char next;

    bufsize = MU_MAX_HEADER;
```

```
    if(NULL == ((char *)*hdr = (char *)malloc(bufsize)))
        return MUNBUF;
    memset(*hdr, 0, bufsize);

    for(i = 0; 1; i ++){

        //extend the buffer for a line of a header
        if(i > bufsize – 1)
            if(NULL == ((char *)*hdr = (char *)realloc(*hdr, bufsize <<=1)))
                return MUNBUF;

        ret = MuIO.Reader(socket, *hdr, 1);
        if(ret == 1){
            if((*hdr)[i] == '\n'){

                //when the line is \r\n，that
                //identicate have reache the end of HEADER
                if(!(i == 0 || (i == 1 && (*hdr)[0] == '\r'))){

                    // we need to check if it
                    //continues on to the other line.
                    ret = MuIO.Peeker(socket, &next, 1);
                    if(ret == 0)
                        return MUEEOF; //end of the file
                    else if(res == -1)
                        return MUERED;//error happened

                    //skip the SP and HT
                    if(next == '\t' || next == ' ')
                        continue;
                }

                (*hdr)[i] = '\0';

                if(i > 0 && (*hdr)[i-1] == '\r')
                    (*hdr)[i-1] = '\0';
                break;
            }
        }else if(ret == 0)
            return MUEEOF;
        else
            return MUERED;
    }
    return MUOK;
}
```

5.10.10. Interface 接口

函数在从套接口接收一行字段时，需要用到在《网络套接口 IO》文档中设计的 IO 函数；

在使用本函数时，需要给本函数传递二个参数，一个是待读取的套接字，另一个是二级指针，用以向调用者返回读取的内容；

对于该二级指针所指向的空间，由调用本函数的程序释放；

[注意]:
1、当读取到报文头尾部时，本函数以 MUOK（GET 或 POST 方式）返回，或是 MUEEOF（HEAD 方式）返回;
*2、当读取到报文尾部时，二级指针返回的**内容长度**为 0;*

5.10.11. Malloc 存储分配

程序中采用默认的大小来申请内存空间，若存储时不足，扩大一倍后，再扩展内存空间；

对于已读取的一行报文，由调用程序负责释放空间；

5.10.12. Restrict 限制

略

5.10.13. Test 测试

略

5.10.14. Unsolve 未解决情况

略

5.11. Mu_GetStatusCode

5.11.1. Name 函数名称

int Mu_GetStatusCode(cosnt char *hdr)

5.11.2. Description 函数名称

从获得的一行报文头中，查找服务器返回的 HTTP 状态码，该状态码以返回值的形式返回；

在函数处理的最后，会将已取得的状态码转换成 int 型；

在程序处理状态码时，会对 HTTP 协议版本信息进行处理，在本程序中，对于小于1.0版本的协议，将以出错返回；

5.11.3. Function 功能

获得服务器返回的 HTTP 状态码
具体的状态码可以查看 RFC2616

5.11.4. Capability 性能

略

5.11.5.Input 输入

hdr：地址指针，指向从报文头中读取的一行；

5.11.6.Output 输出

函数运行的状态码，或是 HTTP 的状态码，包括：
1、无状态码，以 **MUNHST** 返回；
2、HTTP 状态码，以实际正整数值返回；
3、协议版本太低，以 **MUELVR** 返回；

5.11.7. Arithmetic 算法

程序在获取返回的状态码时，首先需要讨论协议的版本号，对于1.0以前的版本，则直接返回错；

待解析的格式如下所示：
HTTP/1.0 200 OK

5.11.8. Process 处理流程

略

5.11.9. Pseudocode 伪代码

```
int Mu_GetStatusCode(const char *hdr)
{
    int mjr, mnr;
    int statuscode;
    char *p=NULL;

    if(strncasecmp(hdr, “HTTP/”, 5))
        return MUNHST;

    hdr += 5;
    p = hdr;

    //get the version of http protocol
    for(mjr = 0; isdigit(*hdr); hdr++)
        mjr = 10*mjr +(*hdr-‘0’);

    if(*hdr != ‘.’ || p == hdr)
        return MUNHST;

    ++hdr;
    p=hdr;
    for(mnr=0; isdigit(*hdr); hdr++)
        mnr= 10 *mnr +(*hdr-‘0’);

    if(*hdr != ‘ ’ || p == hdr)
        return MUNHST;
```

```
    //only process the protocol is greater than 1.0
    if(mjr < 1)
        return MUELVR;

    ++hdr;
    if(!(isdigit(*hdr) &&isdigit(hdr[1]) &&isdigit(hdr[2])))
        return MUNHST;

    statuscode = 100 *(*hdr – '0') + 10 *(hdr[1] – '0') + (hdr[2] – '0');
    return statuscode;
}
```

5.11.10. Interface 接口

函数从一行报文头中获得服务器返回的 HTTP 状态码，以此表明服务器的工作状态；

存储一行报文空间，并不由本函数释放；

5.11.11. Malloc 存储分配

无

5.11.12. Restrict 限制

略

5.11.13. Test 测试

略

5.11.14. Unsolve 未解决情况

略

5.12. Mu_GetContentLen

5.12.1. Name 函数名称

off_t Mu_GetContentLen(const char *hdr)

5.12.2. Description 函数描述

从读取的一行报文头中取出正文长度，获得的长度值以返回值的形式返回，求得的长度字串必须被转换成 off_t 类型；

5.12.3. Function 功能

从报文头中取得正文长度字段

5.12.4. Capability 性能

略

5.12.5. Input 输入

同5.11.5. 中说明

5.12.6. Output 输出

函数运行的状态码，或是报文的正文段长度，包括：
1、无长度值，以 **MUNLEN** 返回；
2、长度值，以 **off_t** 类型返回；

5.12.7. Arithmetic 算法

略

5.12.8. Process 处理流程

服务器以该字段返回所发送的报文中包含的正文段的长度，格式如下：
HTTP/1.1 200 OK
[other http headers]
Content-Length: 10000

从套接口读取的一行报文，在该字段尾部不会包含\r\n 字符，在读取的时候，这类字符已被取消，但是若服务器返回的一个字段由多行组成，在解析的**报文中部（不可能是尾部）**还是会包含\r、\n、\t、空格；

5.12.9. Pseudocode 伪代码

```
off_t Mu_GetContentLen(const char *hdr)
{
    off_t len;

    if(strncasecmp(hdr, “content-length:”, 15))
        return MUNLEN;

    hdr += 15;
    //skip the space or table
    for(; *hdr == ‘ ’ || *hdr == ‘\t’ || *hdr == ‘\r’ || *hdr == ‘\n’; hdr ++)
        ;

    if((NULL == hdr) || (!isdigit(*hdr)))
        return MUNLEN;

    for(len = 0; isdigit(*hdr); hdr++)
        len = 10*len + (*hdr – ‘0’);

    return len;
}
```

5.13. Mu_GetRelocation

5.13.1. Name 函数名称

int Mu_GetRelocation(const char *hdr)

5.13.2. Description 函数描述

取得服务器返回的报文头中包含的重定向地址，Device 必须能解析出该地址，并动态地分配出空间以存储该内容，地址的首地址由全局变量（指针）存储；

[注意]:
1、 在对重定向地址发起请求后，该地址指向的空间必须被释放，地址变量的值必须被设置为 NULL;

5.13.3. Function 功能

取得重定向地址

5.13.4. Capability 性能

略

5.13.5. Input 输入

同5.11.5.中说明

5.13.6. Output 输出

以返回是否取得重定向地址值，包括：
1、取得值，以 **MUOK** 返回；
2、申请空间出错，以 **MUNBUF** 返回；
3、无重定向内容，以 **MUNRLC** 返回；

5.13.7. Arithmetic 算法

重定向时，HTTP 协议规定以状态码**303**返回，并且在报头中包括 Location 字段，该字段中包含了可以重定向的地址；

在这种情况下，服务器返回格式为：
HTTP/1.1. 303 …
[other headers]

Location: http://otherwww.com/file

从套接口读取的一行报文，在该字段尾部不会包含\r\n 字符，在读取的时候，这类字符已被取消，但是若服务器返回的一个字段由多行组成，在解析的**报文中部（不可能是尾部）**还是会包含\r、\n、\t、空格；

5.13.8. Process 处理流程

略

5.13.9. Pseudocode 伪代码

```
char *location = NULL;
int Mu_GetRelocation(const char *hdr)
{
    int p = 0;

    if(strncasecmp(hdr, “location:”, 9))
        return MUNRLC;

    hdr += 9;
    //skip the space
    for(; *hdr == ‘ ’ || *hdr == ‘\t’ || *hdr == ‘\r’ || *hdr == ‘\n’; hdr ++)
        ;

    //store the info
    if(location)
        free(location);

    if((!(p = strlen(hdr))
        || (NULL == ((char *)location = (char *)malloc(p+1))))
        return MUNBUF;

    memset(location, 0, p+1);
    snprintf(location, p, “%s”, hdr);
    return MUOK;
}
```

5.13.10. Interface 接口

略

5.13.11. Malloc 存储分配

函数在存储重定向地址时，使用了全局地址变量来存储地址起始地址，每次存储重定向地址内容时，都需要重新分配内存空间；

[注意]:

1、在重新连接该地址后，应该释放 location 所指向的内存空间;

5.13.12. Restrict 限制

略

5.13.13. Test 测试

略

5.13.14. Unsolve 未解决情况

略

5.14. Mu_GetCookies

5.14.1. Namce 函数名称

int Mu_GetCookies(const char *hdr)

5.14.2. Description 函数描述

从一行报文中取出 Cookies，并存储于动态地址空间；

在下一次请求服务器时，只需要将该 Cookies 添加到报文头，而不需要作其他特殊的处理即可(MUFTAD 项目规定)；

[注意]:

1、当获得一个新的 Cookies 后，需要将原有内容释放，重新获得内存空间;

5.14.3. Function 功能

从报文中获得 Cookies

5.14.4. Capability 性能

略

5.14.5. Input 输入

同5.11.5.中说明

5.14.6.Output 输出

返回函数的运行状态，包括：
1、无错误，以 **MUOK** 返回；
2、无空间，以 **MUNBUF** 返回；
3、无 cookie，以 **MUNCOK** 返回；

5.14.7. Arithmetic 算法

在服务器返回报文中包括 Cookies 字段，报文格式如下所示：
HTTP/1.1 200 OK
[other headers]
Set-Cookie: acct=04382374;domain=.ora.com;Expires=Sun

从套接口读取的一行报文，在该字段尾部不会包含\r\n 字符，在读取的时候，这类字符已被取消，但是若服务器返回的一个字段由多行组成，在解析的**报文中部（不可能是尾部）**还是会包含\r、\n、\t、空格；

5.14.8. Process 处理流程

略

5.14.9. Pseudocode 伪代码

```
char *cookies = NULL;
int Mu_GetCookies(const char *hdr)
{
    int p;

    if(strncasecmp(hdr, “Set-Cookie:”, 11))
        return MUNCOK;

    hdr += 11;
    //skip the space
    for(; *hdr == ‘’ || *hdr == ‘\t’ || *hdr == ‘\r’ || *hdr == ‘\n’; hdr ++)
        ;
    //store the info
    if(cookies)
        free(cookies);

    if( !(p = strstr(hdr))
        || (NULL == ((char *)cookies = (char *)malloc(p+1))))
        return MUNBUF;

    memset(cookies, 0, p+1);
    snprintf(cookies, p, “%s”, hdr);
    return MUOK;
}
```

5.14.10. Interface 接口

略

5.14.11. Malloc 存储分配

函数在存储重定向地址时，使用了全局地址变量来存储地址起始地址，每次存储重定向地址内容时，都需要重新分配内存空间；

[注意]:

1、在重新连接该地址后，应该释放 cookies 所指向的内存空间;

5.15. Mu_ParseHeader

5.15.1. Name 函数名称

int Mu_ParseHeader(int socket, Mu_HttpStatusPtr *Ptr)

5.15.2. Descsription 函数描述

本函数接收服务器返回的报文头，并解析出报头中的相关信息；

5.15.3. Function 功能

解析报文头

5.15.4. Capability 性能

略

5.15.5. Input 输入

socket：套接口，从该套接口接收字符；
Ptr：地址指针，用于传递 HttpStatus 结构体，提供该指针是为了存储解析的结果；

5.15.6. Output 输出

返回函数处理状态，包括：
1、无错误，以 **MUOK** 返回；
2、错误，返回各子函数错误码；

5.15.7. Arithmetci 算法

我们构建的读取函数，每一次只从套接口读取一个字段，针对该内容，我们需要顺序调用解析函数，以解析出报文头中的内容；

另外，解析一个字段后，需要释放存储已解析字段的空间；

5.15.8. Process 处理流程

略

5.15.9. Pseudocode 伪代码

```
int Mu_ParseHeader(int socket，Mu_HttpStatusPtr Ptr)
{
    int ret=MUOK;
    int statusecode = 0;
    int contentlen = 0;
    int relocation = 0;
    int cookies = 0;

    int num_get = 0;
    char *hdr = NULL;

    while(1){
        if((ret = Mu_FetchHeader(socket, &hdr)) < 0){
            if(!hdr)
                free(hdr);
            return ret;
        }

        //if have read the empty line, identicate that
        //we have reach the end of Header
        if(!*hdr){
            free(hdr);
            break;
        }

        if((!statusecode) && (ret = Mu_GetStatusCode(hdr)) > 0) {
            hs->status = ret;
            statusecode = 1;
            free(hdr);
            continue;
        }

        if((!contentlen) && (ret = Mu_GetContentLen(hdr)) > 0){
            hs->bytesexpect = ret;
            hs->bytesleft = ret;
```

```
            contentlen = 1;
            fre(hdr);
            continue;
        }

        if(!relocation){
            if((ret = Mu_GetRelocation(hdr)) < 0) {
                free(hdr);
                if(ret == MUNBUF)
                    return ret;
            }
            else{
                free(hdr);
                relocation = 1;
                continue;
            }
        }

        if(!cookies){
            if((ret = Mu_GetCookies(hdr)) < 0) {
                free(hdr);
                if(ret == MUNBUF)
                    return ret;
            }
            else{
                cookies = 1;
                free(hdr);
                continue;
            }
        }

        free(hdr);
    }
    return MUOK;
}
```

5.15.10. Interface 接口

该函数用于提供接口，以便从服务器返回的报文头中取得相关的参数，调用者需要提供二个参数，一个是待读取的套接口，另一个是用于保存相关参数的结构体 Mu_HttpStat；

并不是所有解析出来的内容均存放于结构 Mu_HttpStat 结构中，针对 Cookie、Relocation 则是存储在动态地址空间，空间的首地址由全局变量指向；

[注意]:

1、cookies 和 relocation 二个全局变量，在存储新的内容时，必须释放二个指针所指向的空间，并且置为 NULL，以防止其他函数释放时产生段错误;

5.15.11. Malloc 存储分配

解析服务器返回的报文头，函数采用逐行的方式读取报文；

函数提供指针 hdr 指向读取的内存空间，在解析完成后，需要释放该空间，否则会发生内存溢出；

5.15.12. Restrict 限制

略

5.15.13. Test 测试

略

5.15.14. Unsolve 未解决情况

现阶段程序只解析 State、Content-Length、Cookies、Relocation 四个字段的内容，其他字段内容暂不支持；

为了扩充该功能，需要添加二个部分：
1、在本函数中，按函数本身添加解析函数入口；
2、完成一个需要的解析函数；

5.16. Mu_PostSignal

5.16.1. Name 函数名称

int Mu_PostSignal(int socket, int xmlfd)

5.16.2. Description 函数描述

该函数用于从 socket 套接字口，将存储在 xmlfd 中的信令文件发送出去。在发送时，使用了 HTTPS 对整个发送过程进行加密；

发送信令时也包括将结束符发送到对端，结束符为**\r\n**；

5.16.3. Function 功能

发送 POST 报文正文段

5.16.4. Capability 性能

略

5.16.5. Input 输入

socket：套接字口，将从此套接口上发送 POST 报文正文段；
xmlfd：存储有信令内容的文件句柄；

5.16.6. Output 输出

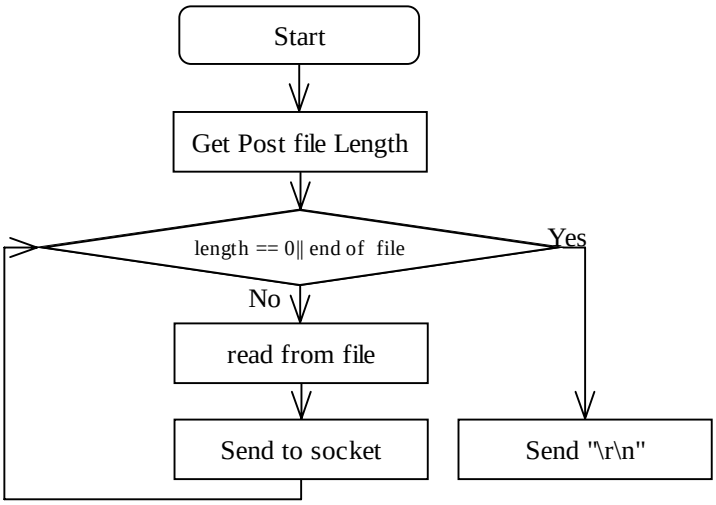
函数发送的状态值，包括：
1、无错误，以 **MUOK** 返回；
2、发送出错，以 **MUESND** 返回；
3、取 POST 内容错，以 **MUEPST** 返回；

5.16.7. Arithmetic 算法

发送 POST 请求信令时，POST 请求报文的报文头由 Mu_BuildQueryPost 函数构建，并由调用者发送至服务器，报文头中已包含分隔字符：**\r\n\r\n**；

POST 正文段的数据包以 xmlfd 中的内容开始，以**\r\n** 结束，最后二个字符以单独的数据包发送至服务器；

5.16.8. Process 处理流程



5.16.9. Pseudocode 伪代码

```
int Mu_PostSignal(int socket, int xmlfd)
{
    char send[2048];
    int readlen = 0;
    off_t totallen;

    memset(send, 0, 2048);
    struct stat info;

    //get the length of the file, which will be post to the server
    if( (!xmlfd) || if(fstat(fileno(xmlfd), &info) == -1)
        return MUEPST;
    totallen = info.st_size;

    //read the contents from the file, write to the socket to server
    while((totallen)&& (readlen = read( xmlfd, send, sizeof(send))) > 0))
        if(MuIO.Writer(socket, send, readlen)<0)
            return MUESND;

        totallen -= readlen;
        readlen = 0;
        memset(send, 0, 2048);
    }
```

```
    if( MuIO.Writer(socket, "\r\n", 2) < 0)
        return MUESND;

    return MUOK;
}
```

5.16.10. Interface 接口

函数从信令文件中读取信令，并发送到相应的套接口，实现过程中需要用到《网络 IO》中实现的 IO 操作函数；

在调用本函数时，需要传递一个调用者正在使用，并可以正常工作的套接口，另外还需要一个非空的文件句柄；

5.16.11. Malloc 存储分配

函数中并未使用动态内存空间，在读取发送信令时，使用了堆栈空间——数组来实现，调用者不需要对该空间进行其他操作；

5.16.12. Restrict 限制

略

5.16.13. Test 测试

略

5.16.14. Unsolve 未解决情况

对最后二个字符，即 POST 报文的结束标记**\r\n**，函数在实现时采用了单独发送的方式；

该方式的实现较之将**\r\n** 附加到信令后面发送，效率要低，但是考虑到实现，并未采用后者；

5.17. Mu_Query

5.17.1. Name 函数名称

int Mu_Query(Mu_DnStatPtr Ptr)

5.17.2. Description 函数描述

该函数根据解析出的地址信息，调用函数构建请求数据包，发送到服务器，并将服务器返回的信息存储到本地；

5.17.3. Function 功能

查询服务器，并获得相关的信息

5.17.4. Capability 性能

略

5.17.5. Input 输入

Ptr：Mu_DnStatPtr 结构体
其成员类型在4.1.2.中说明

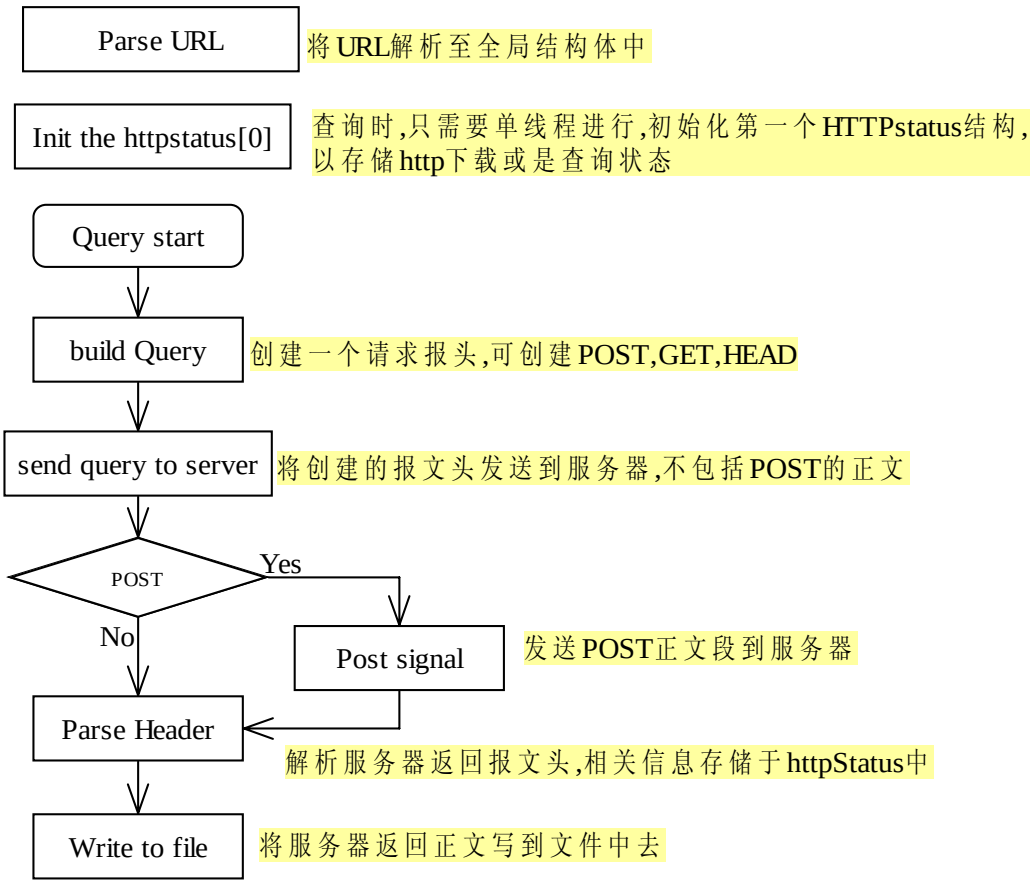
5.17.6.Output 输出

函数运行的状态码，包括：
1、无错误，以 **MUOK** 返回；
2、连接错误，以 **MUECON** 返回；
3、接收错误，以 **MUEQUE** 返回；
4、重连，以 **MUERLC** 返回；

5.17.7.Arithmetic 算法

略

5.17.8. Process 处理流程



5.17.9. Pseudocode 伪代码

```
Mu_HttpStatus hs[MAX_THD];
int Mu_Query(Mu_DnStatPtr Ptr)
{
    char *query = NULL;
    int socket;
    char buffer[2048];
    char *ops;
    int ret;
    int len;
    int count;
    int requerytime = 0;
    int writelen;
    Mu_Build build;
```

```
requery:
    ops = buffer;
    len = 0;
    count = 0;
    writelen = 0;

    Mu_InitHttpStat(&hs[0]); //malloc space to store info,
                             //all members must be init

    gettimeofday(&(hs[temp].starttime, NULL);

    //connect to server, got the socket;
    while(1){
        if(MuIO.Connecter(&socket, mu_url.host, mu_url.port, 30) < 0){
            if(count++ == 10)
                return MUECON;
            fprintf(stdout, "cann't connect to sever! will reconnect…\n");
            sleep(30);
        }
        else{
            fprintf(stdout, "connect SUCCESSFUL!\n");
            hs[temp].threadstat = MU_DNCNT;
            break;
        }
    }

    //create BUIL structure;
    build.method = Ptr->type;
    build.fd = Ptr->xmlfd;
    build.startops = 0;
    build.endops = 0;
    if((ret = Mu_BuildQuery(build, &query) < 0)
        goto error;
    //do build error

    if(MuIO.Writer(socket, query, strlen(query)) < 0){
        goto error;
    }
    if(query){
        free(query);
        query = NULL;
    }

    if(Ptr->type == MU_POST){
        if((ret = Mu_PostSignal(socket, Ptr->xmlfd)) < 0)
```

```
            goto error;
    }

    if((ret = Mu_ParseHeader(socket, &hs[0]) < 0){
        goto error;
    }

    if(CHECK_STATRE(hs[temp].statcode))
        return MUERLC;

    if(!CHECK_STATOK(hs[temp].statcode))
        goto error;

    len = hs[0].bytesleft;
    memset(buffer, 0, 2048);

    while(len){
        if((ret = MuIO.Reader(socket, buffer, 2048)) < 0){
            goto error;
        }
        len -= ret;

        //write to file
        writelen = ret;
        while(writelen){
            if((ret = write(Ptr->fd, ops, writelen)) < 0){
                goto error;
            }
            ops += ret;
            writelen -= ret;
        }

        memset(buffer, 0, 2048);
    }
    write(Ptr->fd, EOF, 1);
    close(socket);
    return MUOK;

error:
    if(ret == MUNBUF)
        return ret;

    requerytime ++;
    if(requerytime == 10)
        return MUEQUE;
```

```
    lseek(Ptr->fd, 0, SEEK_SET);
    if(!socket)
        close(socket);

    if(query){
        free(query);
        query = NULL;
    }
    goto requery;
}
```

5.17.10. Interface 接口

如果查询出错，重新发起查询请求时，调用者必须将 fd，即待记录服务器返回内容的文件句柄，必须被重新定位到文件首部；

当出错或是查询正常结束，函数将关闭套接口，但是并不会关闭 fd 文件句柄；

5.17.11. Malloc 存储分配

请求数据包由 Mu_BuildQuery 构建，该函数会申请动态内存空间，并以二级指针的形式返回，该空间的释放由本函数完成，如函数中的 free(query);

5.17.12. Restrict 限制

略

5.17.13. Test 测试

略

5.17.14. Unsolve 未解决情况

函数对于下载过程中的错误，必须能重新处理，重新下载，现阶段，每个线程可以允许出错10次，否则退出。不过，针对内存空间不足，线程将立即退出，而不会重试；

连接服务器错时，可以尝试10次连接，如果仍然连接不上服务器，函数也将以出错返回，对该错误的处理同上所述；

5.18. Mu_GetLoop

5.18.1. Name 函数名称

int Mu_GetLoop(Mu_DnStatPtr Ptr)

5.18.2. Description 函数描述

该函数初始化下载部分，为多线程下载准备前提条件；
[注意]:
1、该函数并不应用于查询部分;

5.18.3. Function 功能

该函数初始化下载部分

5.18.4. Capability 性能

略

5.18.5. Input 输入

同5.17.5. 中说明

5.18.6. Output 输出

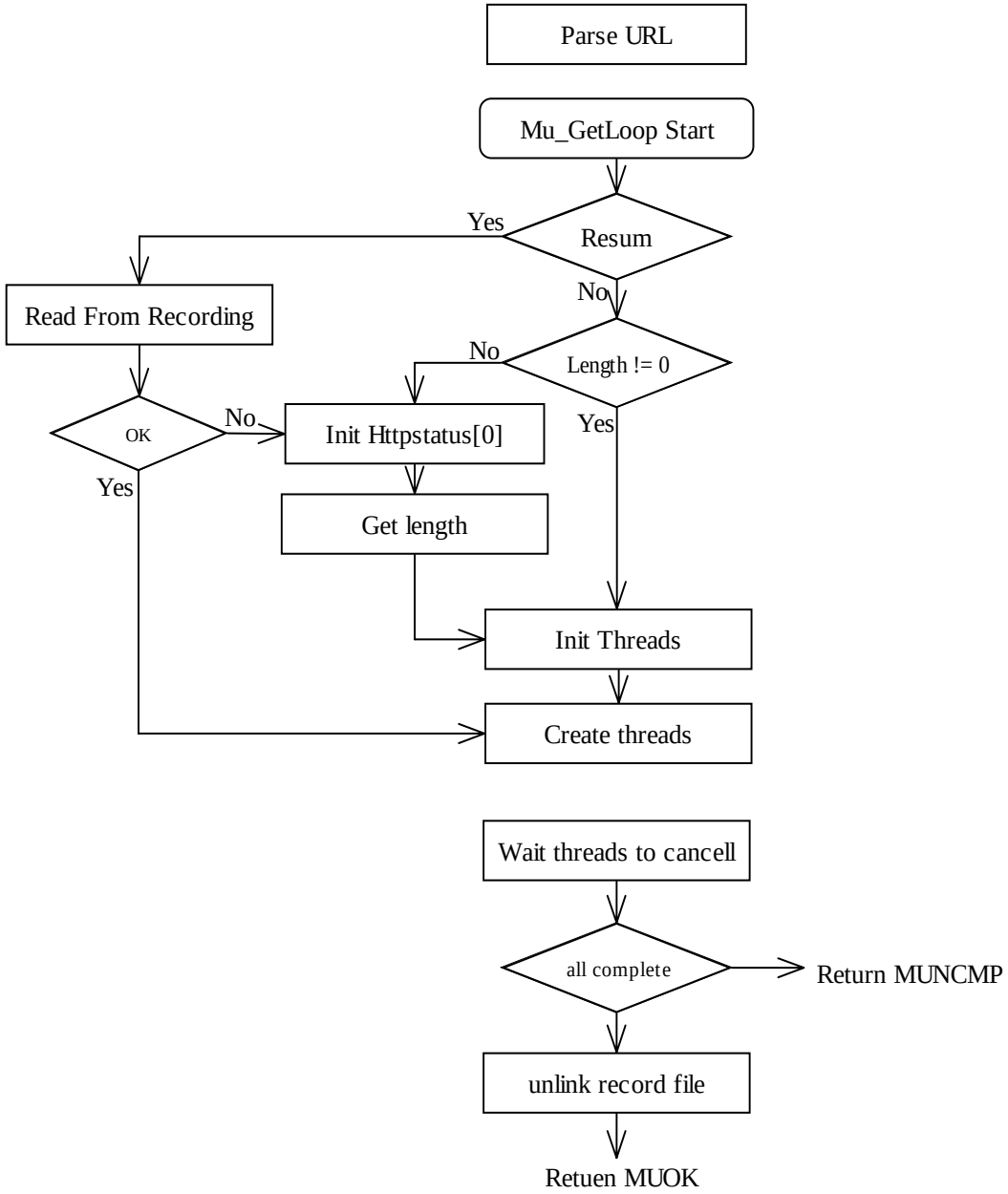
函数返回下载的状态码，包括：
1、无错误，以 **MUOK** 返回；
2、连接错误，以 **MUECON** 返回；
3、发送请求错误，以 **MUESND** 返回；
4、未下载完全，以 **MUNCMP** 返回；
5、删除记录文件出错，以 **MUEDEL** 返回；

6、重定向，以 **MUERLC** 返回；
6、解析报文错误，以解析函数自身的返回值返回；

5.18.7. Arithmetic 算法

略

5.18.8. Process 处理流程



5.18.9. Pseudocode 伪代码

```
Mu_HttpStatus hs[MAX_THD];
int threadnum;
char filename[256];
```

```
int Mu_GetLoop(Mu_DnStatPtr Ptr)
{
	int socket;
	int ret;
	char *query = NULL;

	//get the threads
	if(Ptr->resume){
		//read threads set from recorder
		if(Mu_ReadFromRec(filename) < 0)
			goto newdownload;
	}
	else{
		if(Ptr->length == 0){

newdownload:
			//get the length from server
			Mu_InitHttpStat(&hs[0]);

			//connect to server, got the socket;
			while(1){
				if(MuIO.Connecter(&socket, mu_url.host, mu_url.port, 30) < 0){
					if(count++ == 10)
						return MUECON;
					fprintf(stdout,
						"cann't connect to sever! will reconnect…\n");
					sleep(30);
				}
				else{
					fprintf(stdout, "connect SUCCESSFUL!\n");
					break;
				}
			}

			//create BUIL structure;
			build.method = HEAD;
			build.fd = Ptr->NULL;
			build.startops = 0;
			build.endops = 0;
			if((ret = Mu_BuildQuery(build, &query) < 0)
				return ret ;
				//do build error free(query)

			if(MuIO.Writer(socket, query, strlen(query)) < 0){
				free(query);
```

```
                close(socket);
                return MUESND;
            }
            free(query);

            if((ret = Mu_ParseHeader(socket, &hs[0]) < 0){
                close(socket);
                return ret;
            }

            close(socket);
            Ptr->length = hs[0].btesleft;
        }

        if((ret = Mu_InitThreads(Ptr->length)) <0)
            return ret;

        //copy the startops to sturture HttpStat
        for(i = 0; i < threadnum; i ++)
            hs[i].seekstart = Ptr->startops;
}

//create the threads to download file
for(int i = 0; i < threadnum; i ++)
    pthread_create(&(hs[i].threadnumber),
                        NULL, (void*)Mu_Download, (void*)&i);

//wait threads to cancell
 for(int i=0; i<threadnum; i++)
      pthread_join(hs[i].threadnumber, NULL);

//check the file
for(i = 0; i < threadnum; i ++){
    if(hs[i].statcode == 303)
        return MUERLC;
    if(hs[i].threadstatuse != MU_DNCOMP)
        goto error;
}

if(unlick(tempname) < 0){
    do error;
    return MUEDEL;
}

return MUOK;
```

```
error:
    return MUNCMP;
}
```

5.18.10. Interface 接口

函数初始化下载线程环境，并创建线程，等待线程的结束，并依线程的运行情况返回相应的值；

对于发送，接收错误，函数本身不作重连处理，函数直接以相应的错误码返回，重试由调用者完成；

每个线程下载一个部分，若有线程下载部分内容不完整，函数直接返回 MUECMP 错误，对该错误的处理，由调用者完成；

对于 **MUEDEL** 错误，由调用者完成；

5.18.11. Malloc 存储分配

query，指向 Mu_BuildQuery 函数申请的一段内存空间，该空间存储构建的请求信息，对于 query 指向的内存空间，由本函数释放；

5.18.12. Restrict 限制

略

5.18.13. Test 测试

略

5.18.14. Unsolve 未解决情况

本函数的实现没有考虑线程在下载过程中的异常终止情况，函数实现时一律按正常返回看待；

下载是否完全，由函数判断线程状态值来决定；

5.19. Mu_ReadFromRec

5.19.1. Name 函数名称

int Mu_ReadFromRec(const char * name)

5.19.2. Description 函数描述

本函数从下载过程中的记录文件中读取相关参数，读取的值填充 MuHttpStatus 结构体，从而实现多线程续传；

读取记录的顺序与写记录的顺序相同，否则读取错误；

5.19.3. Function 功能

读取记录，实现续传

5.19.4. Capability 性能

略

5.19.5. Input 输入

name：字符串变量，用于指明记录下载状态的临时文件名；
[注意]:
1、该文件名由调用者指明;
2、在下载完成后，该文件需要被删除，采用 unlink 实现;

5.19.6. Output 输出

返回函数的运行状态，如下所示：
1、无错误，则以 **MUOK** 返回；
2、读取出错，以 **MUERED** 返回；

5.19.7. Arithmetic 算法

读取记录前必须先初始化 Mu_HttpStat 结构体内的成员，以防止错误的赋值；

5.19.8.Process 流程

略

5.19.9.Pseudocode 伪代码

```
int threadnum;
int urllen;
int Mu_ReadFromRec(const char *name)
{
    int statfd;
    int i;
    char url[1024];
    char tempname[256];

    //init httpstats
    for(i = 0; i < MAX_THD; i++)
        Mu_InitHttpStat(&hs[i]);

    memset(tempname, 0, 256);
    snprintf(tempname, “%s%s”, name, “.jw”);

    //open the record file
    if(-1 == (statfd = open(tempname, O_RDONLY)))
        goto error;

    if(read(statfd, &urllen, sizeof(int)) < 0)
        goto error;

    //read the url from the file, and compare the url
    //make sure the address resuming is the same;
    if((read(statfd , url, urllen) <= 0)
        ||((strlen(url) != strlen(mu_url.url))
        ||(!strncasecmp(url, mu_url.url, strlen(url)))){

        fprintf(stdout, “wrong url\n”);
        goto error;
    }
```

```
    //read the numbers of the threads
    if((read(statfd, &threadnum, sizeof(int)) <= 0)
        ||(num > MAX_THD))
        goto error;

    //read the structure of Mu_HttpStatus
    for(i = 0; i<threadnum; i ++){
        if(read(statfd, &hs[i], sizeof(Mu_HttpStatus)) <= 0)
        goto error;
    }

    colse(statfd);
    return MUOK;

error:
    close(statfd);
    return MUERED;
}
```

5.20. Mu_RecToFile

5.20.1. Name 函数名称

```
int Mu_RecToFile(const char *name)
```

5.20.2. Description 函数描述

同函数 Mu_ReadFromRec 相反，该函数将相关（同 Mu_ReadFromRec 中的项相同的项）记录入文件名为 name 的临时文件；

在下载完全后，该文件将会被删除；

5.20.3. Function 功能

记录下载状态；

5.20.4. Capability 性能

略

5.20.5. Input 输入

同5.19.5.中说明

5.20.6. Output 输出

输出函数运行的状态，包括：
1、无错误码，以 **MUOK** 返回；
2、记录出错，以 **MUEREC** 返回；

5.20.7. Arithmetic 算法

略

5.20.8. Process 处理流程

略

5.20.9. Pseudocode 伪代码

```
int urllen;
int Mu_RecToFile(const char *name)
{
    int statfd;
    char filename[256];
    int i;

    memset(filename, 0, 256);
    snprintf(filename, "%s%s", name, ".jw");

    if((statfd = open(name, O_RDWR)) == NULL)
        goto error;

    urllen = strlen(mu_url.url);
```

```
    if(write(statfd, &urllen, sizeof(int)) < 0)
        goto error;

    //write the URL to record file
    if(((write(statfd, mu_url.url, strlen(mu_url.url))) < 0)
        ||((write(statfd, &threadnum, sizeof(int))) < 0))
        goto error;

    //write the structure to file
    for(i = 0; i<threadnum; i ++){
        if(write(statfd, &hs[i], sizeof(Mu_HttpStatus)) < 0)
            goto error;
    }

    fclose(statfd);
    return MUOK;

error:
    fclose(statfd);
    return MUEREC;
}
```

5.21. Mu_InitHttpStat

5.21.1. Name 函数名称

int Mu_InitHttpStat(Mu_HttpStatPtr Ptr)

5.21.2. Description 函数描述

函数初始化一个 Mu_HttpStat 结构，以便于使用该结构记录相应的状态；

在本项目中，HTTP 下载分为单线程和多线程，为了便于断点续传，必须记录各线程的工作状态及待下载文件的读写位置等等信息；

5.21.3. Function 功能

记录一个线程的下载状态

5.21.4. Capability 性能

略

5.21.5. Input 输入

Ptr：Mu_HttpStat 结构指针，是等初始化结构的指针；

5.21.6. Output 输出

该函数无返回值；

5.21.7. Arithmetic 算法

略

5.21.8. Process 处理流程

略

5.21.9. Pseudocode 伪代码

```
void Mu_InitHttpStat(Mu_HttpStatPtr Ptr)
{
    Ptr->threadnumber = 0;
    Ptr->seekstart = 0;
    Ptr->seeknew = 0;

    Ptr->bytesleft= 0;
    Ptr->bytesexcept = 0;

    Ptr->retry = 0;

    Ptr->fd = NULL;
    Ptr->statcode = 0;
    Ptr->threadstatuse = 0;

    Ptr->starttime->tv_sec = 0;
```

```
    Ptr->starttime->tv_usec = 0;
    Ptr->endtime->tv_sec = 0;
    Ptr->endtime->tv_usec = 0;

    Ptr->rates = 0;
    return;
}
```

5.22. Mu_InitThread

5.22.1. Name 函数名称

int Mu_InitThread(off_t length)

5.22.2. Description 函数描述

该函数主要用于划分线程，并将各个线程的信息写入到 Mu_HttpStat 结构数组中，以便各线程在下载时，各自使用互不相同的地址空间和请求不同的下载起始位置；

5.22.3. Function 功能

划分线程

5.22.4. Capability 性能

略

5.22.5. Input 输入

length：一个文件的整体长度；

5.22.6.Output 输出

5.22.7. Arithmetic 算法

函数分线程的算法如下：

1、以最大线程数 MAX_THD 分得各线程下载内容 LEN；

2、若各线程下载内容 LEN 小于单线程最小下载量 MAX_THD_LEN，线程数减一，重复第一步；

若大于最小线程下载量，进行第三步；

3、按 LEN 设置**除最后一线程外**的所有线程的下载量；

4、将余下内容设置为最后一线程的下载量；

5.22.8. Process 处理流程

略

5.22.9. Pseudocode 伪代码

```
#define MAX_THD 10
#define MAX_THD_LEN 0x100 000
int threadnum;

int Mu_IniThread(off_t length)
{
    off_t threadsize;
    off_t temp = 0;
    int tempnum = MAX_THD;
    int num;

    while(tempnum && ((threadsize = (length/tempnum)) < MAX_THD_LEN))
        tempnum --;

    //record the threads' numbers
    if(!tempnum)
        threadnum = 1;
    else
        threadnum = tempnum;

    for(i = 0; i < tempnum; i ++)
        Mu_InitHttpStat(&(hs.[i]);

    //set the NO. of    the bytes, every thread will download…
    for(i = 0; i < tempnum - 1; i ++){
```

```
        //hs[i].bytesleft = threadsize;
        hs[i].bytesexpect = threadsize;
        hs[i].seeknew = threadsize * i;
    }

    //get ride of the last thread, in order to make sure the file is complete
    //hs[i].bytesleft = length – (threadsize * i);
    hs[i].bytesexpect = length – (threadsize * i );
    hs[i].seeknew = threadsize * i;

    return MUOK;
}
```

5.22.10. Interface 接口

该函数在实现时，需要明确二点：

1、hs[i].bytesexpect 为该线程所必须下载的总的字节数，该字段在该函数中被初始化后，不会被其他任何函数修改；

2、hs[i].bytesleft 为该线程还没有下载完成的内容长度，该字段可以在该部分被设置，也可在 GET 请求服务器时，解析服务器的报文头得到；

但是该字段会依下载内容而更改；

[注意]:

1、下载时，可能需要对 bytesleft，bytesexpect 进行比较;

3、hs[i].seeknew 为该线程在下载其需要内容时，请求服务器开始传输节目内容的开始点，在构建请求报文头时，需要该值；

该值单独即可表明请求下载的开始点；

写入文件时，也需要用到该值，**hs[i].seeknew + hs[i].seekstart** 即可得到当前线程写入文件的起始位置；

[注意]:

1、各线程单独维护一个文件句柄，因此也就单独维护一个文件的偏移量;

5.22.11. Malloc 存储分配

略

5.22.12. Restrict 限制

略

5.22.13. Test 测试

略

5.22.14. Unsolve 未解决情况

略

5.23. Mu_Download

5.23.1. Name 函数名称

void Mu_Download(int *num)

5.23.2. Description 函数描述

本函数用于下载节目内容，下载内容的大小由各结构体 Mu_Httpstat 决定，同时，将内容写入结构体中指定的文件及指定的位置；

下载过程中，需要将各自的下载状态记入结构体中的 threadstatus 成员中，以便于所有线程结束后，判别下载是否完成；

对于已下载完成的线程，需要立即退出；

5.23.3. Function 功能

下载节目

5.23.4. Capability 性能

略

5.23.5. Input 输入

num：int 类型，用于指明 Mu_Httpsta 数组中的位置，该参数由 Pthread_create 以指针的形式传入；

5.23.6. Output 输出

无

5.23.7. Arithmetic 算法

略

5.23.8. Process 处理流程

略

5.23.9. Pseudocode 伪代码

```
char filename[256];
pthread_mutex_t lock;
#define CHECK_STATOK(x) ((x) >= 200) && ((x) < = 299)
#define CHECK_STATRE(x) ((x) == 303)

int Mu_Download(int *num)
{
    //check the thread wethere it's compelete
    if(hs[temp].threadstatuse == MU_DNCMP)
        return;

    int socket = 0;
    Mu_Build build;
    int temp = *num;
    BOOL connected;
    int boolprint = 0;
    char buffer[4096];
    off_t startops= hs[temp].seeknew; //back up the seeknew
    int revlen;
    int count;
```

```
    int redown;
    unsigned long timeuse;

redownload:
    connected = FALSE;
    boolprintf = 0;
    revlen = 0;
    count = 0;
    redown = 0;

    //get the start time of the download
    //use the lock to make sure the synchronization
    pthread_mutex_lock(&lock);
    gettimeofday(&(hs[temp].starttime, NULL);
    pthread_mutex_unlock(&lock);

    //connect to server, got the socket;
    while(1){
        if(MuIO.Connecter(&socket, mu_url.host, mu_url.port, 30) < 0){
            if(count++ == 10)
                return;
            fprintf(stdout, "cann't connect to sever! will reconnect…\n");
            sleep(30);
        }else{
            fprintf(stdout, "connect SUCCESSFUL!\n");

            //change the statuse of thread
            pthread_mutex_lock(&lock);
            hs[temp].threadstat = MU_DNCNT;
            pthread_mutex_unlock(&lock);

            break;
        }
    }

    //make the query
    build.method = MU_GET;
    build.fd = NULL;
    build.startops = hs[i].seeknew;
    build.endops = build.seeknew + hs[i].bytesexcept - 1;

    if(((ret = Mu_BuildQuery(build, &query)) < 0)
        ||((ret = MuIO.Writer(socket, query, strlen(query)) < 0)
        ||((ret = Mu_ParseHeader(socket, &(hs[temp])) < 0))
        goto error;
```

```
if(CHECK_STATRE(hs[temp].statcode))
    return MUERLC;

if(!CHECK_STATOK(hs[temp].statcode))
    goto error;

if(query){
    free(query);
    query = NULL;
}

if(hs[temp].bytesexcept != hs[temp].bytesleft)
    return;

memset(buffer, 0, 4096);
if(-1 == (hs[temp].fd = open(filename, O_WRONLY|O_CREAT, 0644))
    return;

lseek(hs[temp].fd, hs[temp].seekstart + hs[temp].seeknew, SEEK_SET);

pthread_mutex_lock(&lock);
hs[temp].threadstat = MU_DNRCV;
pthread_mutex_unlock(&lock);

while(hs[temp].bytesleft){

    //record counting
    boolprintf ++ ;
    if(boolprint == 10)
        boolprint = 0;

    if((ret= MuIO.Reader(socket, buffer, 4096) < 0)
        goto error;
    revlen = ret;

    //write to file
    writelen = ret;
    while(writelen){
        if((ret = write(hs[temop].fd, buffer, writelen) < 0)
            goto error;

        startops += ret;
        writelen -= ret;
    }
```

```
            pthread_mutex_lock(&lock)
            hs[temp].bytesleft -= revlen;
            hs[temp].seeknew += revlen;

            //calculate the rate of downloading
            gettimeofday(&(hs[temp].teimend), NULL);
            timeuse = 1000 000 *(hs[temp].starttime.tv_sec –hs[temp].endtime.tv_sec)
                        + hs[temp].starttime.tv_usec –hs[temp].endtime.tv_usec
            timeuse /= 1000 000;
            hs[temp].rate = revlen / timeuse;

            //record to file
            if(boolprint == 0)
                Mu_RecToFile(filename);
            pthread_mutex_unlock(&lock);

            memset(buffer, 0, 4096);
        }

        pthread_mutex_lock(&lock);
        hs[temp].threadstat = MU_DNCMP;
        pthread_mutex_unlock(&lock);

        close(socket);
        close(hs[temp].fd);
        free(query);
        return;

error:
        fprintf(stdout, “redownload the files ….\n”);
        redown ++;

        if(ret == MUNBUF) //make sure thre MUNBUF is not -1
            return;

        //free query buffer
        if(query){
            free(query);
            query = NULL;
        }

        //close fd
        if(hs[temp].fd){
            close(hs[temp].fd);
            hs[temp].fd = 0;
```

```
    }

    //close socket
    if(socket){
        close(socket);
        socket = 0;
    }

    if(redown == 10)
        return;
    else{
        pthread_mutex_lock(&lock);
        hs[temp].threadstat = MU_RECNT;
        pthread_mutex_unlock(&lock);
        goto redownload;
    }
}
```

5.23.10. Interface 接口

该函数是多线程运行的主体，因此该函数必须是可重入的；

函数对于下载过程中的错误，必须能重新处理，重新下载，现阶段，每个线程可以允许出错10次，否则退出。不过，针对内存空间不足，线程将立即退出，而不会重试；

在下载过程中，线程需要相应地改变其状态值，以便于主线程确定一线程是否完成下载；

该函数中对文件的读写，采用了 LINUX 库函数，而不是标准 IO 读写函数；

5.23.11. Malloc 存储分配

接收时需要用到的记录空间由 buffer 指明，该内存空间位于堆栈中，在线程退出时，空间自动释放；

请求报文使用的空间为动态内存，需要函数显示地释放该空间；

5.23.12. Restrict 限制

略

5.23.13. Test 测试

略

5.23.14. Unsolve 未解决情况

在函数实现时，使用锁来同步，该操作对程序运行效率有一定的影响，linuxdown 程序，对此则没有采用此操作，linuxdown 作者认为该状态下的不同步是可以接受的；

但是本函数实现时仍采用锁来同步；

5.24. Mu_HttpPlus

5.24.1. Name 函数名称

int Mu_HttpPlus(const char *url, int method, Mu_DnStatPtr Ptr)

5.24.2. Description 函数描述

函数接收调用者的相关参数，解析 URL，将 URL 相关信息存储到 mu_url 全局变量中；

根据 Mu_DnStatPtr 结构中的成员值，采用不同的方式构建请求报文，查询，交互信令，或是下载节目等；

该函数是下载模块向用户提供的唯一下载接口；

5.24.3. Function 功能

解析 URL，根据参数处理 HTTP 请求

5.24.4. Capability 性能

略

5.24.5. Input 输入

url：URL 地址，该地址必须完整有效，有效性由调用者保证；
method：标注调用者使用 HTTP 是进行查询还是下载；
[注意]:
1、查询一律使用单线程进行处理，而下载全部使用多线程;

Ptr：Mu_DnStatPtr 结构，该结构中的相关成员表明调用者想进行的 HTTP 请求类型，包括查询和下载；

5.24.6. Output 输出

函数的处理状态，包括：
1、无错误，以 **MUOK** 返回；
2、连接错误，以 **MUECON** 返回；
3、接收错误，以 **MUEQUE** 返回；
4、发送请求错误，以 **MUESND** 返回；
5、未下载完全，以 **MUNCMP** 返回；
6、删除记录文件出错，以 **MUEDEL** 返回；
7、解析 URL 错，以 **MUEURL** 返回；
8、取空间信息出错，以 **MUEFSY** 返回；
9、可用空间小于2%，以 **MUNSPA** 返回；
10、调用 HTTP 类型错误，以 **MUEMEH** 返回
11、解析报文错误，以解析函数自身的返回值返回；

5.24.7. Arithmetci 算法

略

5.24.8.Process 处理流程

略

5.24.9. Pseudocode 伪代码

```
int Mu_HttpPlus(cosnt char *url, int method, Mu_DnStatPtr Ptr)
{
    int ret;
    struct statfs fs_info;
    int space_rate;

retry:
    if((ret = Mu_ParseUrl(url)) < 0)
        return MUEURL;

    switch(method){
        case MU_QUERY:
            if((ret = Mu_Query(Ptr)) < 0){
                //process the relocation
                if(ret == MUERLC){
                    url = relocate;
                    goto retry;
                }else
                    goto error;
            }
        case MU_DOWNLOAD:
            if((ret = Mu_GetLoop(Ptr)) < 0){
                //process the relocation
                if(ret == MUERLC){
                    url = relocate;
                    goto retry;
                }else
                    goto error;
            }
        default:
            return MUEMEH;
    }
    return MUOK;

error:
    if(statfs("/", &fs_info)!=0){
        fprintf(stdout, " unable to stat the file system for writing \n");
        return MUEFSY;
    }

    if((space_rate =
            (int)(100 *((double)fs_info.f_bavail / (double)fs_info.f_blocks))) < 2)
```

```
        return MUENSP;
    else
        return ret;
}
```

6. Appendix 附录

6.1 Mu_GetBasicAuth

```
char *Mu_GetBasicAuth(const char *user, const char *passwd)
{
    char *p1, *p2;
    char *ret = NULL;
    int len = strlen(user) + strlen(passwd) + 1;
    int b64len = 4 * ((len + 2) / 3);

    p1 = kmalloc(sizeof(char) * len + 1);
    sprintf(p1, "%s:%s", user, passwd);
    p2 = kmalloc(sizeof(char) * b64len + 1);

    /*Encode username:passwd to base 64 */
    Mu_Base64Encode(p1, p2, len);
    ret = kmalloc(b64len + 11);
    free(p1);
    free(p2);

    return ret;
}
```

6.2. Mu_Base64Encode

```
void Mu_Base64Encode(const char *s, char *store, int length)
{
    /* Conversion table.    */
    char tbl[64] = {
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
    'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
    'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
    'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
    'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
    'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
```

```
    'w', 'x', 'y', 'z', '0', '1', '2', '3',
    '4', '5', '6', '7', '8', '9', '+', '/'
     };
     int i;
     unsigned char *p = (unsigned char *) store;

     /* Transform the 3x8 bits to 4x6 bits, as required by base64.    */
     for (i = 0; i < length; i += 3)
     {
    *p++ = tbl[s[0] >> 2];
    *p++ = tbl[((s[0] & 3) << 4) + (s[1] >> 4)];
    *p++ = tbl[((s[1] & 0xf) << 2) + (s[2] >> 6)];
    *p++ = tbl[s[2] & 0x3f];
    s += 3;
     }
     /* Pad the result if necessary...    */
     if (i == length + 1)
    *(p - 1) = '=';
     else if (i == length + 2)
    *(p - 1) = *(p - 2) = '=';
     /* ...and zero-terminate it.    */
     *p = '\0';
}
```

6.3. Mu_SslInit

```
SSL_CTX ssl_ctx;
#define SSLVER23
int Mu_SslInit ()
{
    SSL_METHOD *meth;

    if (ssl_ctx)
        /* The SSL has already been initialized. */
        return MUESSL;

    /* Init the PRNG.    If that fails, bail out.    */
    init_prng ();
    if (RAND_status () != 1){
        fprintf (stdout,"Could not seed PRNG; consider using --random-file.\n");
        goto error;
    }
```

```
    SSL_library_init ();
    SSL_load_error_strings ();
    SSLeay_add_all_algorithms ();
    SSLeay_add_ssl_algorithms ();

# if defined (SSLVER2)
     meth = SSLv2_client_method ();
#elif defined (SSLVER3)
     meth = SSLv3_client_method ();
#elif defined (SSLVER1)
     meth = TLSv1_client_method ();
#else defined (SSLVER23)
      meth = SSLv23_client_method ();
#else
    abort();
#endif

    ssl_ctx = SSL_CTX_new (meth);
    if (!ssl_ctx)
        goto error;
    Mu_CheckCA();

    return MUOK;

    error:
    if (ssl_ctx)
        SSL_CTX_free (ssl_ctx);
    return MUESSL;
}
```

6.4. Mu_CheckCA

```
int Mu_CheckCA()
{
    return MUOK;
}
```