MINIFTPD 项目手册

（第 1 版）

C++教程网 www.cppcourse.com

Mr.J 著

目录

第1章 FTP 协议

1.1 FTP 简介

文件传输协议 FTP（File Transfer Protocol,由 RFC 959 描述）。

FTP 工作在 TCP/IP 协议族的应用层，其传输层使用的是 TCP 协议，它是基于客户/服务器模式工作的。

1.2 FTP 支持的文件类型

ASCII 码文件，这是 FTP 默认的文本格式

EBCDIC 码文件，它也是一种文本类型文件，用 8 位代码表示一个字符，该文本文件在传输时要求两端都使用 EBCDIC 码

图象(Image)文件，也称二进制文件类型，发送的数据为连续的比特流，通常用于传输二进制文件

本地文件，字节的大小有本地主机定义，也就是说每一字节的比特数由发送方规定

1.3 FTP 文件的数据结构

文件结构，这是 FTP 默认的方式，文件被认为是一个连续的字节流，文件内部没有表示结构的信息

记录结构，该结构只适用于文本文件（ASCII 码或 EBCDIC 码文件）。记录结构文件是由连续的记录构成的

页结构，在 FTP 中，文件的一个部分被称为页。当文件是由非连续的多个部分组成时，使用页结构，这种文件称为随机访问文件。每页都带有页号发送，以便收方能随机地存储各页

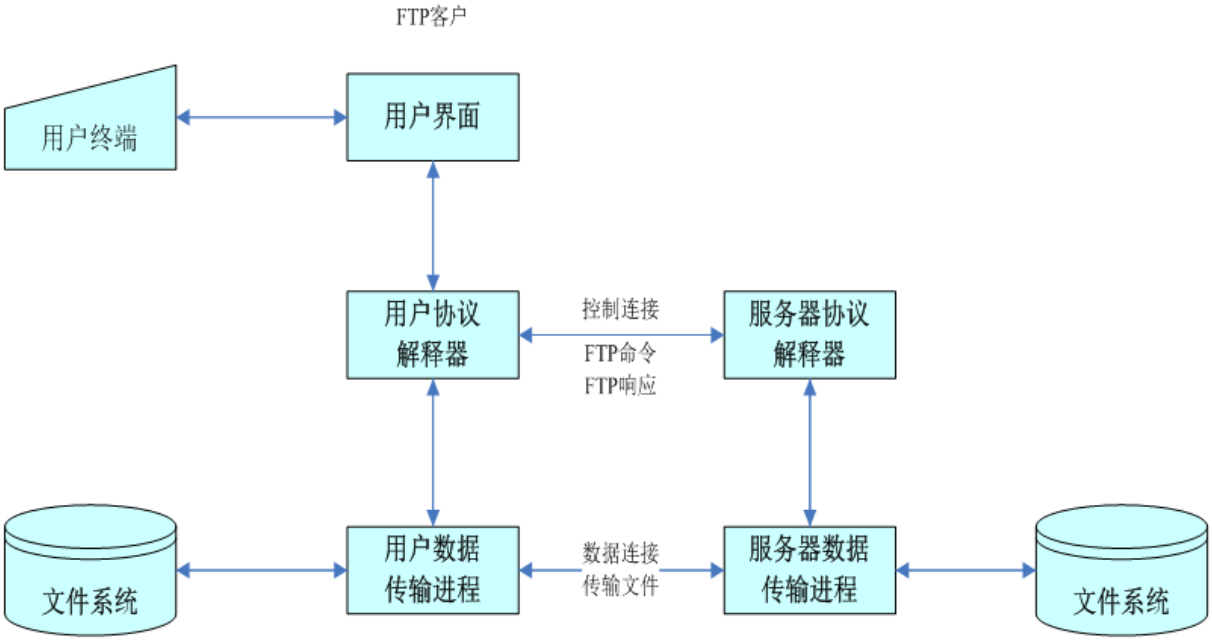
1.4 文件的传输方式

流方式，这是支持文件传输的默认方式，文件以字节流的形式传输。

块方式，文件以一系列块来传输，每块前面都带有自己的头部。头部包含描述子代码域（8 位）和计数域（16 位），描述子代码域定义数据块的结束标志登内容，计数域说明了数据块的字节数

压缩方式，用来对连续出现的相同字节进行压缩，现在已很少使用

1.5 FTP 工作原理



1.5.1 启动 FTP

在客户端，通过交互式的用户界面，客户从终端上输入启动 FTP 的用户交互式命令

1.5.2 建立控制连接

客户端 TCP 协议层根据用户命令给出的服务器 IP 地址，向服务器提供 FTP 服务的 21 端口（该端口是 TCP 协议层用来传输 FTP 命令的端口）发出主动建立连接的请求。服务器收到请求后，通过 3 次握手，就在进行 FTP 命令处理的用户协议解释器进程和服务器协议解释器进程之间建立了一条 TCP 连接。

以后所有用户输入的 FTP 命令和服务器的应答都由该连接进行传输，因此把它叫做控制连接。

1.5.3 建立数据连接

当客户通过交互式的用户界面，向 FTP 服务器发出要下载服务器上某一文件的命令时，该命令被送到用户协议解释器。

1.5.4 关闭 FTP

当客户发出退出 FTP 的交互式命令时，控制连接被关闭，FTP 服务结束

1.6 FTP 命令

命令类型	命令	功能说明
访问控制命令	USER	服务器上的用户名。
	PASS	用户口令。
	CWD 或 XCWD	改变工作目录。
	CDUP 或 XCUP	回到上一层目录（父目录）
	QUIT	退出
	ACCT	
	SMNT	
	REIN	
传输参数命令	PORT	数据端口，主要向服务器发送客户数据连接的端口，格式为 PORT h1,h2,h3,h4,p1,p2,其中 32 位的 IP 地址用 h1,h2,h3,h4 表示，16 位的 TCP 端口号用 p1,p2 表示。
	PASV	此命令要求服务器数据传输进程在随机端口上监听，进入被动接收请求的状态。
	TYPE	文件类型，可指定 ASCII 码，二进制等。

	STRU	文件结构
	MODE	传输模式
服务命令	RETR	获得文件
	STOR	保存文件，向服务器传输文件。如果文件已存在，原文件将被覆盖，如果文件不存在，则新建文件。
	APPE	与 STOR 功能类似，但如果文件在指定路径已存在，则把数据附加到原文件尾部，如果不存在，则新建一个文件。
	LIST	列目录详细清单
	NLST	列出名字列表
	REST	重新开始，参数代表服务器要重新开始的那一点，它并不传送文件，而是略过指定点前的数据，此命令后应该跟其他要求文件传输的 FTP 命令。
	ABOR	异常终止。此命令通知服务中止以前的 FTP 命令和与之相关的数据传输。如果先前的操作已完成，则没有动作，返回 226;如果没有完成，返回 426，再返回 226。
	PWD 或 XPWD	打印当前目录
	MKD 或 XMKD	新建目录
	RMD 或 XRMD	删除目录
	DELE	删除文件
	RNFR，RNTO	重命名
	SITE CHMOD	修改权限
	SYST	获取系统信息
	FEAT	服务器特性
	SIZE	获得文件大小
	STAT	返回服务器状态
	NOOP	该命令不指定任何动作，只是要求服务器返回 OK 响应。
	HELP	帮助
	STOU	暂不实现

	ALLO	暂不实现

1.7 FTP 应答

1.7.1 FTP 应答格式

服务器通过控制连接发送给客户的 FTP 应答，由 ASCII 码形式的 3 位数字和一行文本提示信息组成，它们之间用一个空格分割。

应答信息的每行文本以回车<CR>和换行<LF>对结尾。如果需要产生一条多行的应答，第一行在 3 位数字应答代码之后包含一个连字符“一”，而不是空格符；最后一行包含相同的 3 位数字应答代码，后跟一个空格符

1.7.2 FTP 应答作用

确保在文件传输过程中的请求和正在执行的动作保持一致

保证用户程序总是可以得到服务器的状态信息，用户可以根据收到的状态信息对服务器是否正常执行了有关操作进行判定。

1.7.3 FTP 应答数字含义

第一位数字标识了响应是好，坏或者未完成。

应答	说明
1yz	预备状态
2yz	完成状态
3yz	中间状态
4yz	暂时拒绝状态
5yz	永久拒绝状态

第二位数响应大概是发生了什么错误（比如，文件系统错误，语法错误）

应答	说明
x0z	语法 - 这种响应指出了语法错误。给出的命令不存在、没有被实现、或多余。
x1z	信息 - 对于请求信息的响应，比如对状态或帮助的请求。
x2z	连接 - 关于控制连接和数据连接的响应。
x3z	身份验证和帐户 - 对登陆过程和帐户处理的响应。
x4z	未使用
x5z	文件系统 - 请求传输时服务器文件系统的状态或其他文件系统动作状态。

第三位为第二位数字更详细的说明

如：

500 Syntax error, command unrecognized. （语法错误，命令不能被识别）

可能包含因为命令行太长的错误。

501 Syntax error in parameters or arguments. （参数语法错误）

502 Command not implemented. （命令没有实现）

503 Bad sequence of commands. （命令顺序错误）

504 Command not implemented for that parameter. （没有实现这个命令参数）

1.7.4 FTP 应答示例

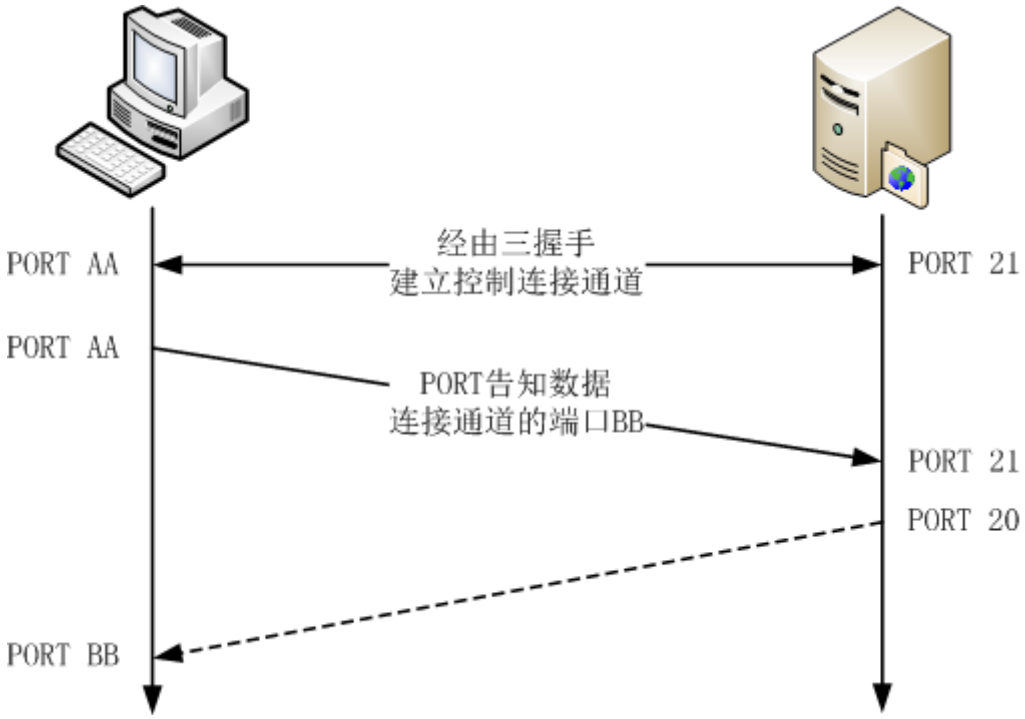
```
#define FTP_DATACONN            150

#define FTP_NOOPOK              200
#define FTP_TYPEOK              200
#define FTP_PORTOK              200
#define FTP_EPRTOK              200
#define FTP_UMASKOK             200
#define FTP_CHMODOK             200
#define FTP_EPSVALLOK           200
#define FTP_STRUOK              200
```

```
#define FTP_MODEOK          200
#define FTP_PBSZOK          200
#define FTP_PROTOK          200
#define FTP_OPTSOK          200
#define FTP_ALLOOK          202
#define FTP_FEAT            211
#define FTP_STATOK          211
#define FTP_SIZEOK          213
#define FTP_MDTMOK          213
#define FTP_STATFILE_OK     213
#define FTP_SITEHELP        214
#define FTP_HELP            214
#define FTP_SYSTOK          215
#define FTP_GREET           220
#define FTP_GOODBYE         221
#define FTP_ABOR_NOCONN     225
#define FTP_TRANSFEROK      226
#define FTP_ABOROK          226
#define FTP_PASVOK          227
#define FTP_EPSVOK          229
#define FTP_LOGINOK         230
#define FTP_AUTHOK          234
#define FTP_CWDOK           250
#define FTP_RMDIROK         250
#define FTP_DELEOK          250
#define FTP_RENAMEOK        250
#define FTP_PWDOK           257
#define FTP_MKDIROK         257

#define FTP_GIVEPWORD       331
#define FTP_RESTOK          350
#define FTP_RNFROK          350

#define FTP_IDLE_TIMEOUT    421
#define FTP_DATA_TIMEOUT    421
#define FTP_TOO_MANY_USERS  421
```

```
#define FTP_IP_LIMIT                421
#define FTP_IP_DENY                 421
#define FTP_TLS_FAIL                421
#define FTP_BADSENDCONN             425
#define FTP_BADSENDNET              426
#define FTP_BADSENDFILE             451

#define FTP_BADCMD                  500
#define FTP_BADOPTS                 501
#define FTP_COMMANDNOTIMPL          502
#define FTP_NEEDUSER                503
#define FTP_NEEDRNFR                503
#define FTP_BADPBSZ                 503
#define FTP_BADPROT                 503
#define FTP_BADSTRU                 504
#define FTP_BADMODE                 504
#define FTP_BADAUTH                 504
#define FTP_NOSUCHPROT              504
#define FTP_NEEDENCRYPT             522
#define FTP_EPSVBAD                 522
#define FTP_DATATLSBAD              522
#define FTP_LOGINERR                530
#define FTP_NOHANDLEPROT            536
#define FTP_FILEFAIL                550
#define FTP_NOPERM                  550
#define FTP_UPLOADFAIL              553
```

1.8 FTP 两种工作模式

1.8.1 主动模式



PORT模式工作过程

1 **客户端向服务器端发送 PORT 命令**

客户端创建数据套接字

客户端绑定一个临时端口

客户端在套接字上监听

将 IP 与端口格式化为 h1,h2,h3,h4,p1,p2

2 **服务器端以 200 响应**

服务器端解析客户端发过来的 IP 与端口暂存起来，以便后续建立数据连接

3 **客户端向服务器端发送 LIST**

服务器端检测在收到 LIST 命令之前是否接收过 PORT 或 PASV 命令

如果没有接收过，则响应 425 Use PORT or PASV first.

如果有接收过，并且是 PORT，则服务器端创建数据套接字（bind 20 端口），调用 connect 主动连接客户端 IP 与端口，从而建立了数据连接

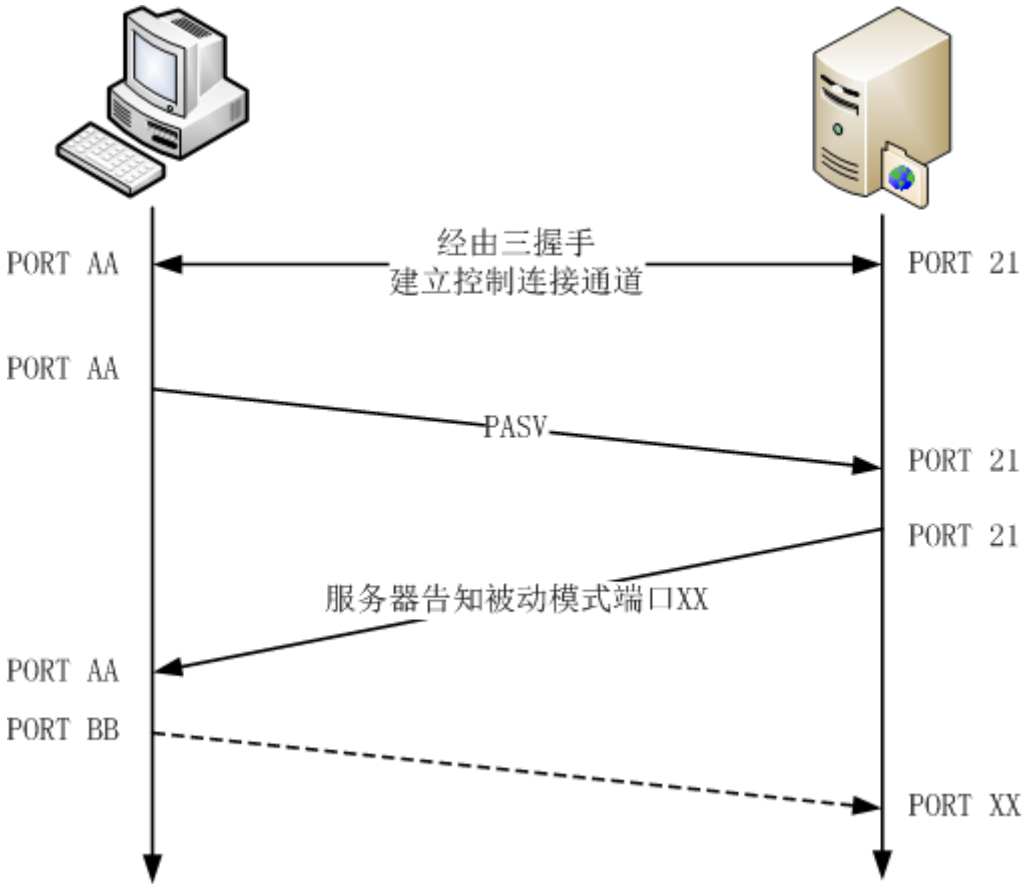
4 **服务发送 150 应答给客户端，表示准备就绪，可以开始传输了。**

5 **开始传输列表**

6 **服务器发送 226 应答给客户端，表示数据传输结束。**

传输结束，服务器端主动关闭数据套接字。

1.8.2 被动模式



PASV模式工作过程

1 **客户端向服务器端发送 PASV 命令**

2 **服务器端以 227 响应**

服务器端创建监听套接字

服务器端绑定一个临时端口

服务器在套接字上监听

将 IP 与端口格式化为 h1,h2,h3,h4,p1,p2 响应给客户端，以便客户端发起数据连接

3 **客户端向服务器端发送 LIST**

服务器端检测在收到 LIST 命令之前是否接收过 PORT 或 PASV 命令

如果没有接收过，则响应 425 Use PORT or PASV first.

如果有接收过，并且是 PASV，则调用 accept 被动接受客户端的连接，返回已连接套接字，从而建立了数据连接。

4 **服务发送 150 应答给客户端，表示准备就绪，可以开始传输了。**

5 **开始传输列表**

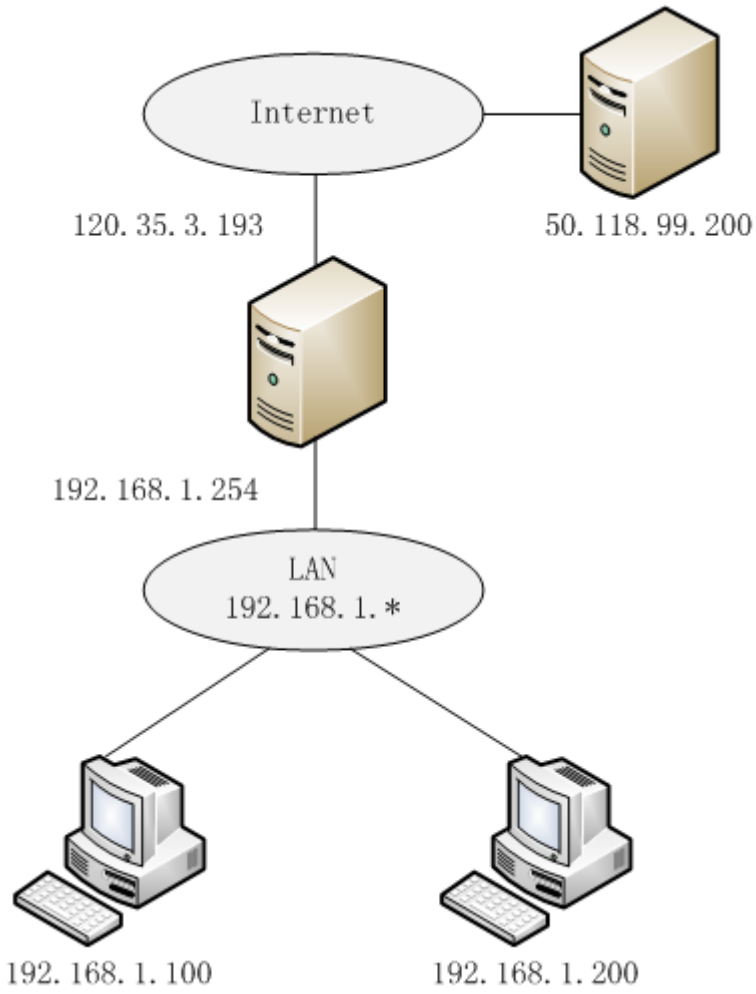
6 **服务器发送 226 应答给客户端，表示数据传输结束。**

传输结束，客户端主动关闭数据套接字。

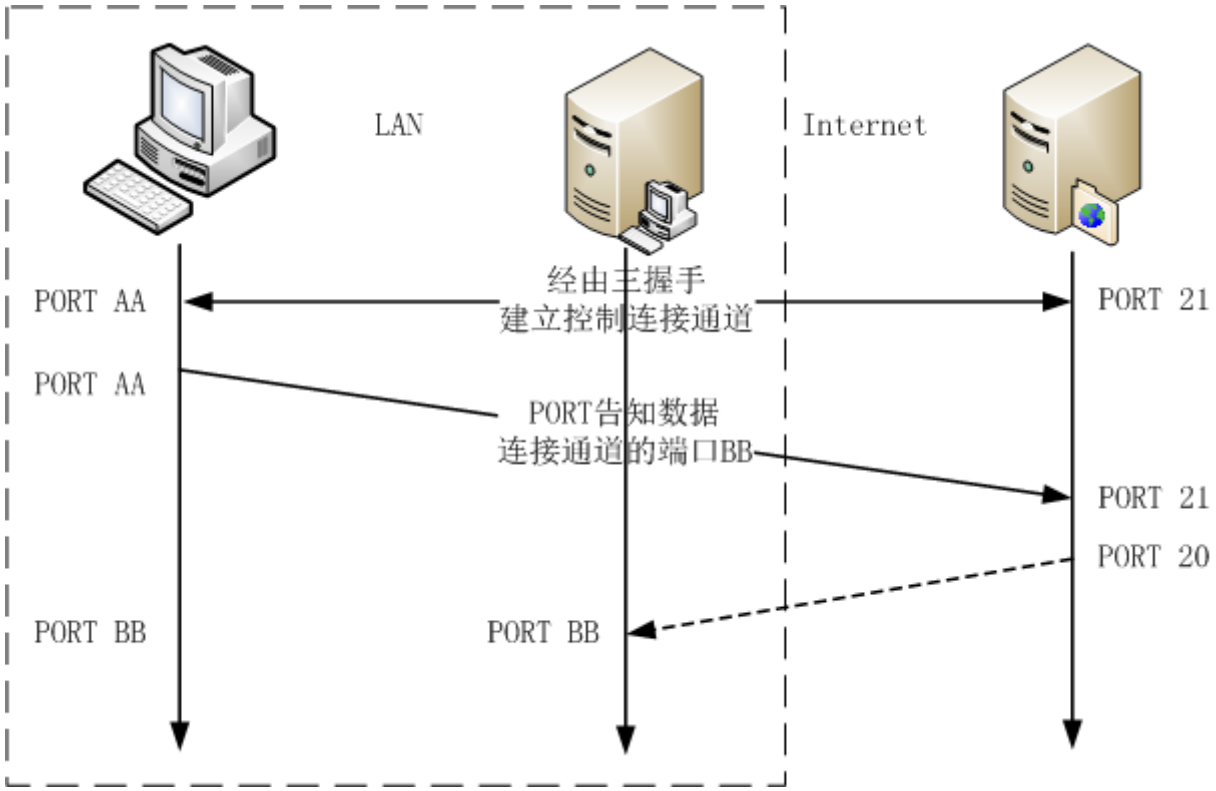
1.8.3 NAT 或防火墙对主被动模式的影响

1.8.3.1什么是 NAT

NAT 的全称是（Network Address Translation），通过 NAT 可以将内网私有 IP 地址转换为公网 IP 地址。一定程度上解决了公网地址不足的问题。



1.8.3.2FTP 客户端处于 NAT 或防火墙之后的主动模式



建立控制连接通道

因为 NAT 会主动记录由内部发送外部的连接信息，而控制连接通道的建立是由客户向服务器端连接的，因此这一条接可以顺利地建立起来。

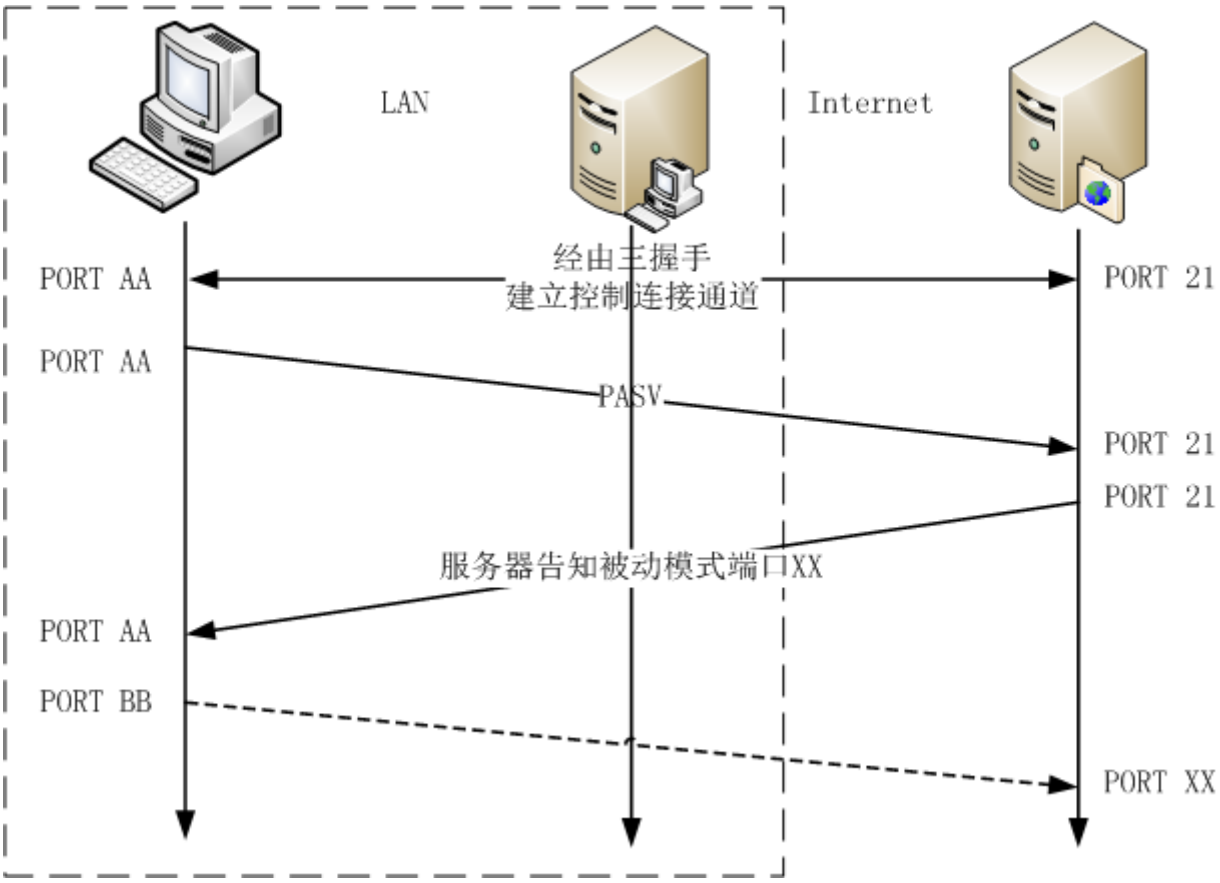
客户端与服务器端数据连接建立时的通知

客户端先启用 PORT BB 端口，并通过命令通道告知 FTP 服务器，且等待服务器端的主动连接。

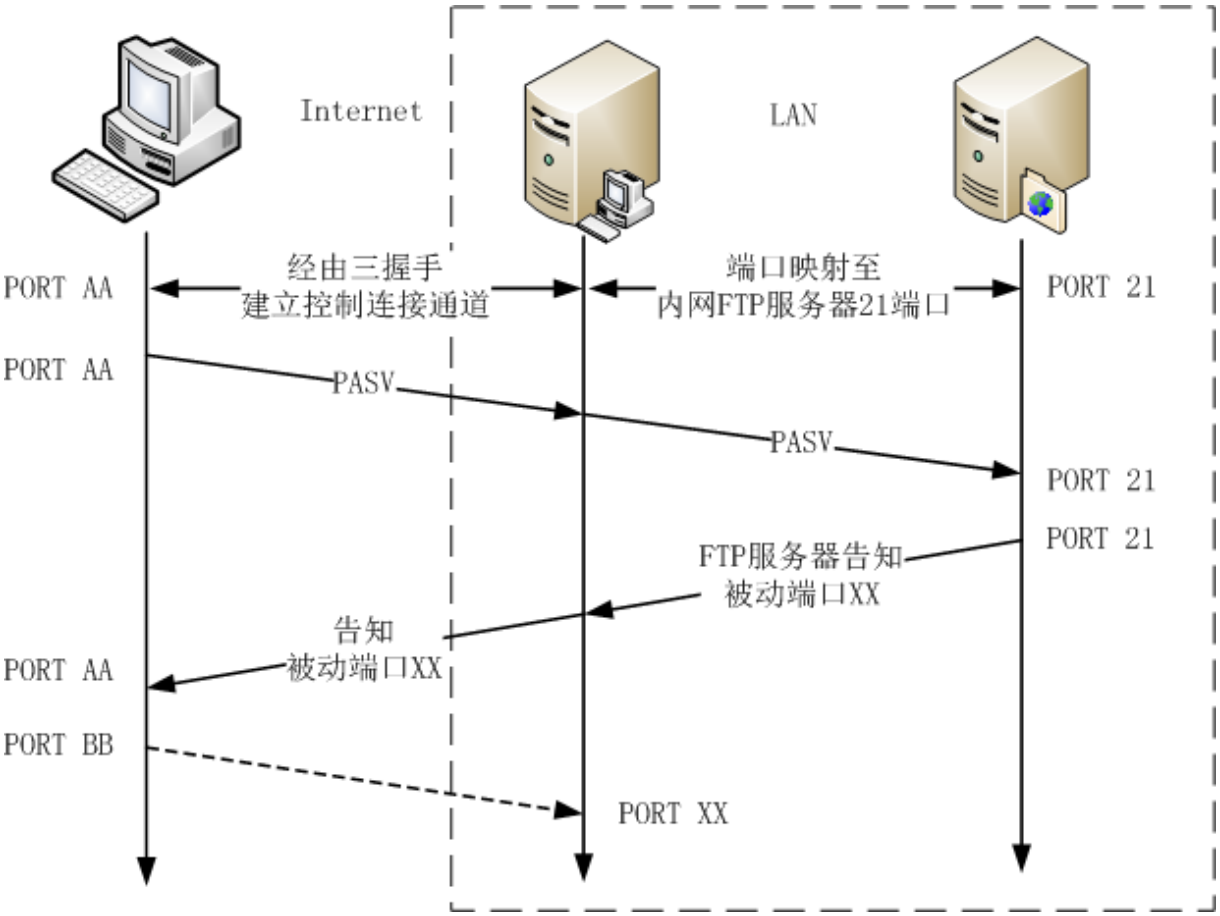
服务器主动连接客户端

由于通过 NAT 转换之后，服务器只能得知 NAT 的地址并不知道客户端的 IP 地址，因此 FTP 服务器会以 20 端口主动地向 NAT 的 PORT BB 端口发送主动连接请求，但 NAT 并没有启用 PORT BB 端口，因而连接被拒绝。

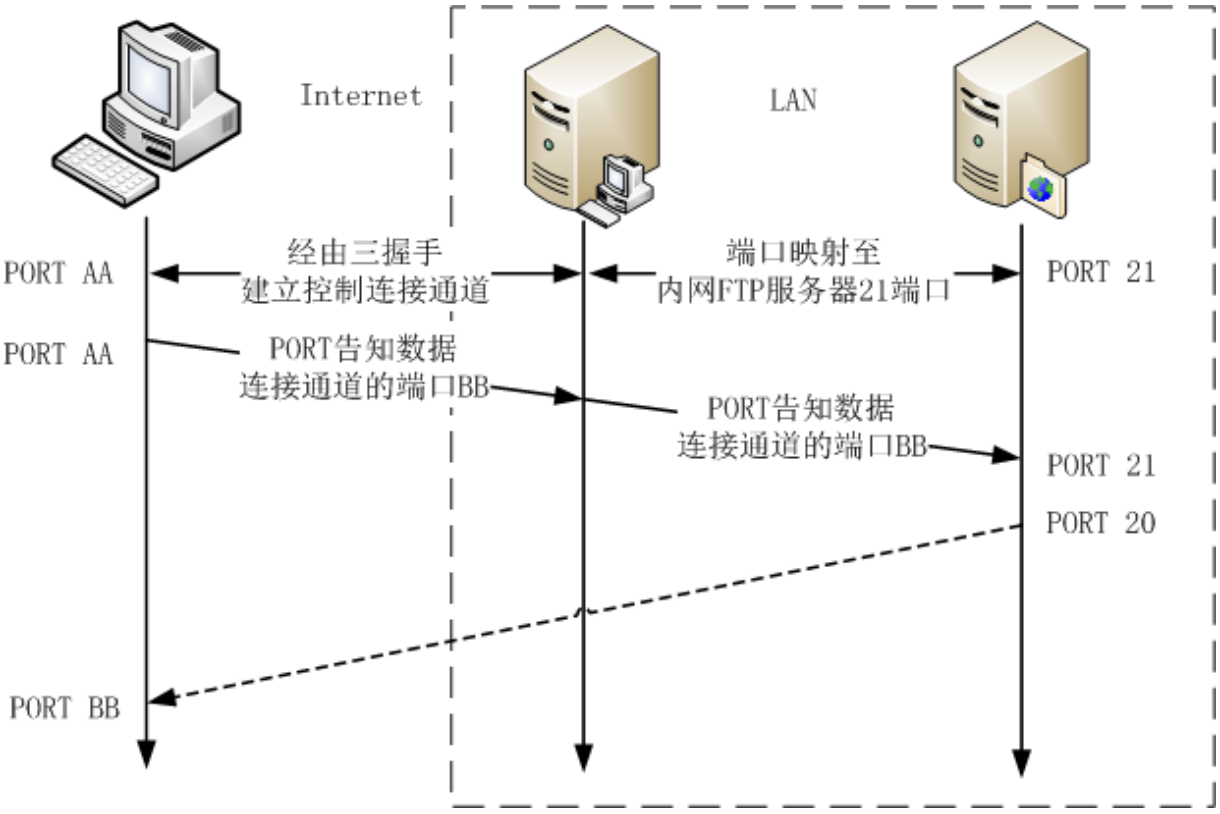
1.8.3.3FTP 客户端处于 NAT 或防火墙之后的被动模式



1.8.3.4FTP 服务器处于 NAT 或防火墙之后的被动模式



1.8.3.5FTP 服务器处于 NAT 或防火墙之后的主动模式


Internet
LAN
PORT AA
经由三握手
建立控制连接通道
端口映射至
内网FTP服务器21端口
PORT 21
PORT AA
PORT告知数据
连接通道的端口BB
PORT告知数据
连接通道的端口BB
PORT 21
PORT 20
PORT BB

第2章 项目需求

2.1 ftp 命令列表

命令类型	命令	功能说明
访问控制命令	USER	服务器上的用户名。
	PASS	用户口令。
	CWD 或 XCWD	改变工作目录。
	CDUP 或 XCUP	回到上一层目录（父目录）
	QUIT	退出
	ACCT	暂不实现
	SMNT	暂不实现
	REIN	暂不实现
传输参数命令	PORT	数据端口，主要向服务器发送客户数据连接的端口，格式为 PORT h1,h2,h3,h4,p1,p2,其中 32 位的 IP 地址用 h1,h2,h3,h4 表示，16 位的 TCP 端口号用 p1,p2 表示。
	PASV	此命令要求服务器数据传输进程在随机端口上监听，进入被动接收请求的状态。
	TYPE	文件类型，可指定 ASCII 码，二进制等。
	STRU	文件结构
	MODE	传输模式
服务命令	RETR	获得文件
	STOR	保存文件，向服务器传输文件。如果文件已存在，原文件将被覆盖，如果文件不存在，则新建文件。
	APPE	与 STOR 功能类似，但如果文件在指定路径已存在，

		则把数据附加到原文件尾部，如果不存在，则新建一个文件。
	LIST	列目录详细清单
	NLST	列出名字列表
	REST	重新开始，参数代表服务器要重新开始的那一点，它并不传送文件，而是略过指定点前的数据，此命令后应该跟其他要求文件传输的 FTP 命令。
	ABOR	异常终止。此命令通知服务中止以前的 FTP 命令和与之相关的数据传输。如果先前的操作已完成，则没有动作，返回 226;如果没有完成，返回 225。
	PWD 或 XPWD	打印当前目录
	MKD 或 XMKD	新建目录
	RMD 或 XRMD	删除目录
	DELE	删除文件
	RNFR，RNTO	重命名
	SITE CHMOD	修改权限
	SYST	获取系统信息
	FEAT	服务器特性
	SIZE	获得文件大小
	STAT	返回服务器状态
	NOOP	该命令不指定任何动作，只是要求服务器返回 OK 响应。
	HELP	帮助
	STOU	暂不实现
	ALLO	暂不实现

2.2 参数配置

2.3 空闲断开

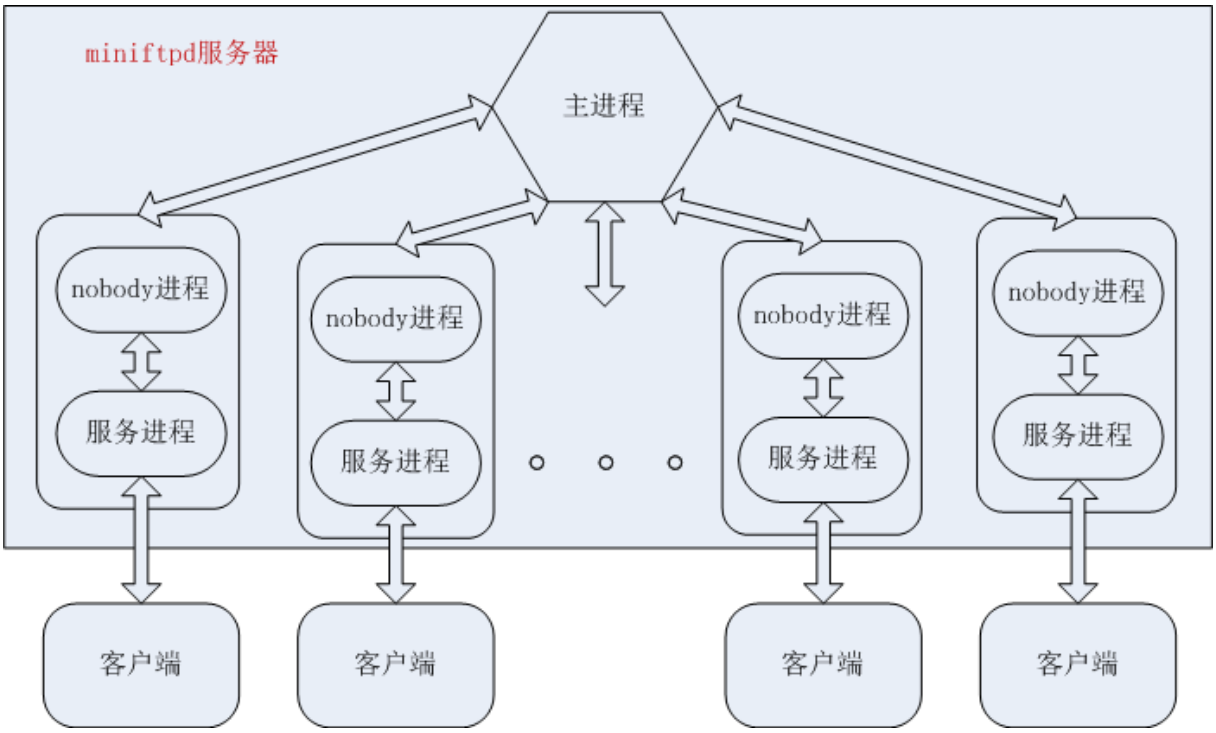
2.4 限速

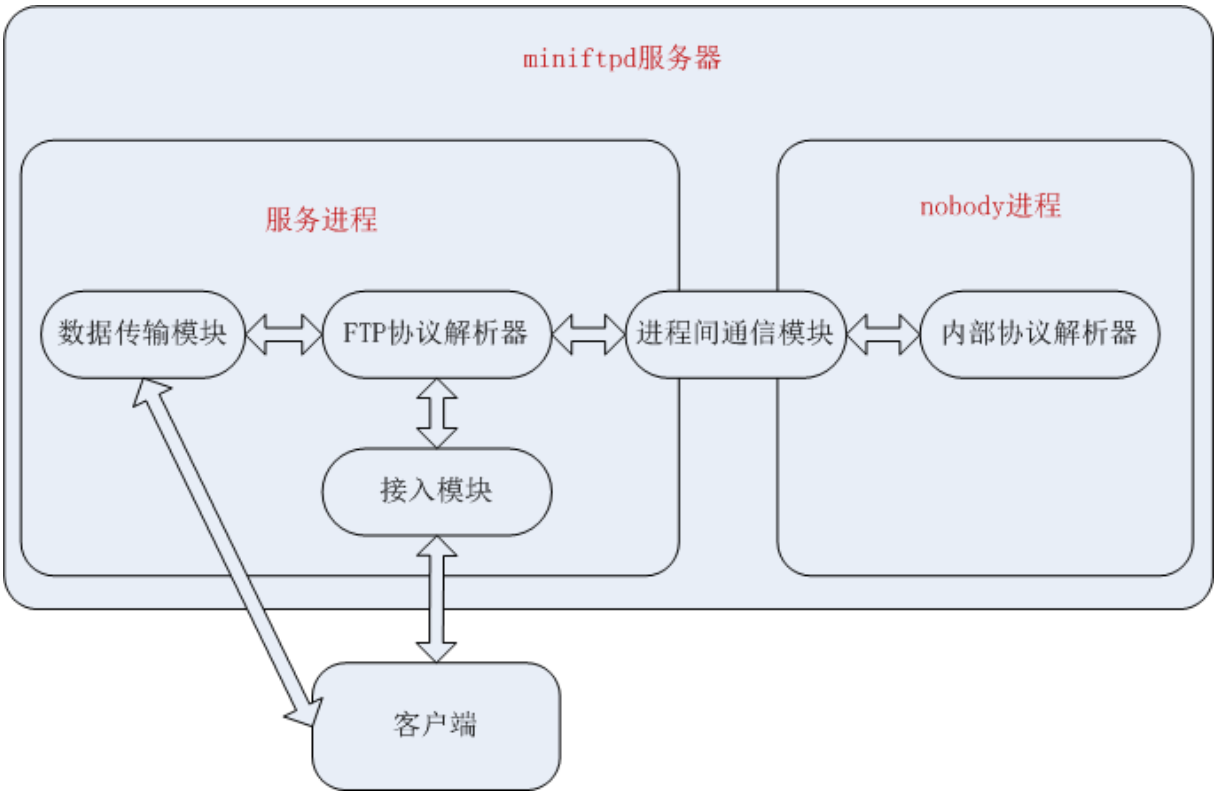
2.5 连接数限制

2.6 断点续载与断点续传

第3章 系统设计

3.1 系统逻辑结构





3.2 字符串工具封装

函数	说明
void str_trim_crlf(char *str);	去除\r\n
void str_split(const char *str , char *left, char *right, char c);	字符串分割
int str_all_space(const char *str);	判断是否全是空白字符
void str_upper(char *str);	字符串转化为大写格式
long long str_to_longlong(const char *str);	将字符串转换为 long long
unsigned int str_octal_to_uint(const char *str);	将字符串（八进制）转化为无符串整型

3.3 参数配置模块设计

配置项变量	说明
int tunable_pasv_enable = 1;	是否开启被动模式
int tunable_port_enable = 1;	是否开启主动模式
unsigned int tunable_listen_port = 21;	FTP 服务器端口
unsigned int tunable_max_clients = 2000;	最大连接数
unsigned int tunable_max_per_ip = 50;	每 IP 最大连接数
unsigned int tunable_accept_timeout = 60;	accept 超时间
unsigned int tunable_connect_timeout = 60;	connect 超时间
unsigned int tunable_idle_session_timeout = 300;	控制连接超时时间
unsigned int tunable_data_connection_timeout = 300;	数据连接超时时间
unsigned int tunable_local_umask = 077;	掩码
unsigned int tunable_upload_max_rate = 0;	最大上传速度
unsigned int tunable_download_max_rate = 0;	最大下载速度
const char *tunable_listen_address;	FTP 服务器 IP 地址

函数	说明
void parseconf_load_file(const char *path);	加载配置文件
void parseconf_load_setting(const char *setting);	将配置项加载到相应的变量

配置文件中的配置项与配置项变量对应关系表

static struct parseconf_bool_setting
{
const char *p_setting_name;
int *p_variable;
}
parseconf_bool_array[] =
{
{ "pasv_enable", &tunable_pasv_enable },

```
    { "port_enable", &tunable_port_enable },
    { NULL, NULL }
};
```

```
static struct parseconf_uint_setting
{
    const char *p_setting_name;
    unsigned int *p_variable;
}
parseconf_uint_array[] =
{
    { "listen_port", &tunable_listen_port },
    { "max_clients", &tunable_max_clients },
    { "max_per_ip", &tunable_max_per_ip },
    { "accept_timeout", &tunable_accept_timeout },
    { "connect_timeout", &tunable_connect_timeout },
    { "idle_session_timeout", &tunable_idle_session_timeout },
    { "data_connection_timeout", &tunable_data_connection_timeout },
    { "local_umask", &tunable_local_umask },
    { "upload_max_rate", &tunable_upload_max_rate },
    { "download_max_rate", &tunable_download_max_rate },
    { NULL, NULL }
};
```

```
static struct parseconf_str_setting
{
    const char *p_setting_name;
    const char **p_variable;
}
parseconf_str_array[] =
{
    { "listen_address", &tunable_listen_address },
    { NULL, NULL }
};
```

3.4 FTP 命令映射的实现

函数	说明
static void do_user(session_t *sess);	
static void do_pass(session_t *sess);	
static void do_cwd(session_t *sess);	
static void do_cdup(session_t *sess);	
static void do_quit(session_t *sess);	
static void do_port(session_t *sess);	
static void do_pasv(session_t *sess);	
static void do_type(session_t *sess);	
static void do_stru(session_t *sess);	
static void do_mode(session_t *sess);	
static void do_retr(session_t *sess);	
static void do_stor(session_t *sess);	
static void do_appe(session_t *sess);	
static void do_list(session_t *sess);	
static void do_nlst(session_t *sess);	
static void do_rest(session_t *sess);	
static void do_abor(session_t *sess);	
static void do_pwd(session_t *sess);	
static void do_mkd(session_t *sess);	
static void do_rmd(session_t *sess);	
static void do_dele(session_t *sess);	
static void do_rnfr(session_t *sess);	
static void do_rnto(session_t *sess);	
static void do_site(session_t *sess);	
static void do_syst(session_t *sess);	
static void do_feat(session_t *sess);	
static void do_size(session_t *sess);	
static void do_stat(session_t *sess);	
static void do_noop(session_t *sess);	
static void do_help(session_t *sess);	

FTP 命令与命令处理函数对应表

```
typedef struct ftpcmd
{
    const char *cmd;
    void (*cmd_handler)(session_t *sess);
} ftpcmd_t;

static ftpcmd_t ctrl_cmds[] = {
    /* 访问控制命令 */
    {"USER",    do_user  },
    {"PASS", do_pass  },
    {"CWD",     do_cwd   },
    {"XCWD",    do_cwd   },
    {"CDUP",    do_cdup  },
    {"XCUP",    do_cdup  },
    {"QUIT", do_quit  },
    {"ACCT",    NULL     },
    {"SMNT",    NULL     },
    {"REIN", NULL     },
    /* 传输参数命令 */
    {"PORT",    do_port  },
    {"PASV",    do_pasv  },
    {"TYPE",    do_type  },
    {"STRU",    do_stru  },
    {"MODE",    do_mode },

    /* 服务命令 */
    {"RETR",    do_retr  },
    {"STOR",    do_stor  },
    {"APPE",do_appe  },
    {"LIST", do_list  },
    {"NLST",    do_nlst  },
    {"REST",do_rest  },
    {"ABOR",    do_abor  },
```

```
    {"\377\364\377\362ABOR", do_abor},
    {"PWD",     do_pwd  },
    {"XPWD",    do_pwd  },
    {"MKD",     do_mkd  },
    {"XMKD",    do_mkd  },
    {"RMD",     do_rmd  },
    {"XRMD",    do_rmd  },
    {"DELE",    do_dele  },
    {"RNFR",    do_rnfr  },
    {"RNTO",    do_rnto  },
    {"SITE", do_site   },
    {"SYST",do_syst   },
    {"FEAT",do_feat },
    {"SIZE", do_size   },
    {"STAT", do_stat   },
    {"NOOP",    do_noop  },
    {"HELP",    do_help  },
    {"STOU",    NULL    },
    {"ALLO",    NULL    }
};
```

3.5 内部进程间通信模块设计

privsock.c 模块

```
// FTP 服务进程向 nobody 进程请求的命令
#define PRIV_SOCK_GET_DATA_SOCK      1
#define PRIV_SOCK_PASV_ACTIVE        2
#define PRIV_SOCK_PASV_LISTEN        3
#define PRIV_SOCK_PASV_ACCEPT        4

// nobody 进程对 FTP 服务进程的应答
#define PRIV_SOCK_RESULT_OK          1
#define PRIV_SOCK_RESULT_BAD         2
```

函数	说明
void priv_sock_init(session_t *sess);	初始化内部进程间通信通道
void priv_sock_close(session_t *sess);	关闭内部进程间通信通道
void priv_sock_set_parent_context(session_t *sess);	设置父进程环境
void priv_sock_set_child_context(session_t *sess);	设置子进程环境
void priv_sock_send_cmd(int fd, char cmd);	发送命令（子->父）
char priv_sock_get_cmd(int fd);	接收命令（父<-子）
void priv_sock_send_result(int fd, char res);	发送结果（父->子）
char priv_sock_get_result(int fd);	接收结果（子<-父）
void priv_sock_send_int(int fd, int the_int);	发送一个整数
int priv_sock_get_int(int fd);	接收一个整数
void priv_sock_send_buf(int fd, const char *buf, unsigned int len);	发送一个字符串
void priv_sock_recv_buf(int fd, char *buf, unsigned int len);	接收一个字符串
void priv_sock_send_fd(int sock_fd, int fd);	发送文件描述符
int priv_sock_recv_fd(int sock_fd);	接收文件描述符

privparent.c 模块

函数	说明
static void privop_pasv_get_data_sock(session_t *sess)	
static void privop_pasv_active(session_t *sess)	
static void privop_pasv_listen(session_t *sess)	
static void privop_pasv_accept(session_t *sess)	

PRIV_SOCK_GET_DATA_SOCK 请求包：

序号	数据项	长度	说明
1	PRIV_SOCK_GET_DATA_SOCK	1	请求 PORT 模式数据套接字
2	端口	4	端口
3	IP 地址	不定	IP 地址

PRIV_SOCK_GET_DATA_SOCK 应答包：

序号	数据项	长度	说明
1	PRIV_SOCK_RESULT_OK 或 PRIV_SOCK_RESULT_BAD	1	如果为 PRIV_SOCK_RESULT_OK 需要应答 PORT_FD
2	PORT_FD	4	应答 PORT 模式套接字

PRIV_SOCK_PASV_ACTIVE 请求包：

序号	数据项	长度	说明
1	PRIV_SOCK_PASV_ACTIVE	1	判定是否处于 PASV 模式

PRIV_SOCK_PASV_ACTIVE 应答包：

序号	数据项	长度	说明
1	ACTIVE	4	0 或者 1

PRIV_SOCK_PASV_LISTEN 请求包：

序号	数据项	长度	说明
1	PRIV_SOCK_PASV_LISTEN	1	获取 PASV 模式监听端口

PRIV_SOCK_PASV_LISTEN 应答包：

序号	数据项	长度	说明
1	LISTEN_PORT	4	应答监听端口

PRIV_SOCK_PASV_ACCEPT 请求包：

序号	数据项	长度	说明
1	PRIV_SOCK_PASV_ACCEPT	1	请求 PASV 模式数据套接字

PRIV_SOCK_PASV_ACTIVE_ACCEPT 应答包：

序号	数据项	长度	说明
1	PRIV_SOCK_RESULT_OK 或 PRIV_SOCK_RESULT_BAD	1	如果为 PRIV_SOCK_RESULT_OK 需要应答 PASV_FD
2	PASV_FD	4	应答 PASV 模式已连接套接字

3.6 断点续载与断点续传

3.6.1 断点续载

REST pos
REST 命令指令断点位置

RETR

3.6.2 断点续传

REST 与 STOR 组合实现断点续传
APPE 实现断点续传

3.7 限速实现

限速的关键是睡眠，如果发现当前传输速度超过最大传输速度就让进程睡眠。

传输速度 = 传输字节数 / 传输时间;

IF 当前传输速度 > 最大传输速度 THEN
    睡眠时间 = (当前传输速度 / 最大传输速度 – 1) * 当前传输时间;

速度 1 / 速度 2 = 时间 2 / 时间 1
速度 1 / 速度 2 - 1 = 时间 2 / 时间 1 -1 = (时间 2 - 时间 1) / 时间 1
(时间 2 - 时间 1) = (速度 1 / 速度 2 – 1) *时间 1

3.8 空闲断开实现

3.8.1 控制连接空闲断开

首先是安装信号 SIGALRM，并启动定时闹钟

如果在闹钟到来之前没有收到任何命令，则在 SIGALRM 信号处理程序中关闭控制连接，并给客户 421 Timeout.的响应，并且退出会话。

3.8.2 数据连接空闲断开

如果当前处于数据传输的状态，那么即使控制连接通道空闲(在空闲时间内没有收到任何客户端的命令)也不应该退出会话。实现方法，只需要将先前设定的闹钟关闭掉。

数据连接通道建立了，但是在一定时间没有传输数据，那么也应该将整个会话断开。

在传输数据之前安装信号 SIGALRM，并启动闹钟。

在传输数据的过程中，如果收到 SIGALRM 信号：

如果 sess->data_process = 0，则给客户端超时的响应 421 Data timeout. Reconnect. Sorry.，并且退出会话。

如果 sess->data_process = 1，将 sess->data_process = 0; 重新安装信号 SIGALRM，并启动闹钟。

3.9 ABOR 实现

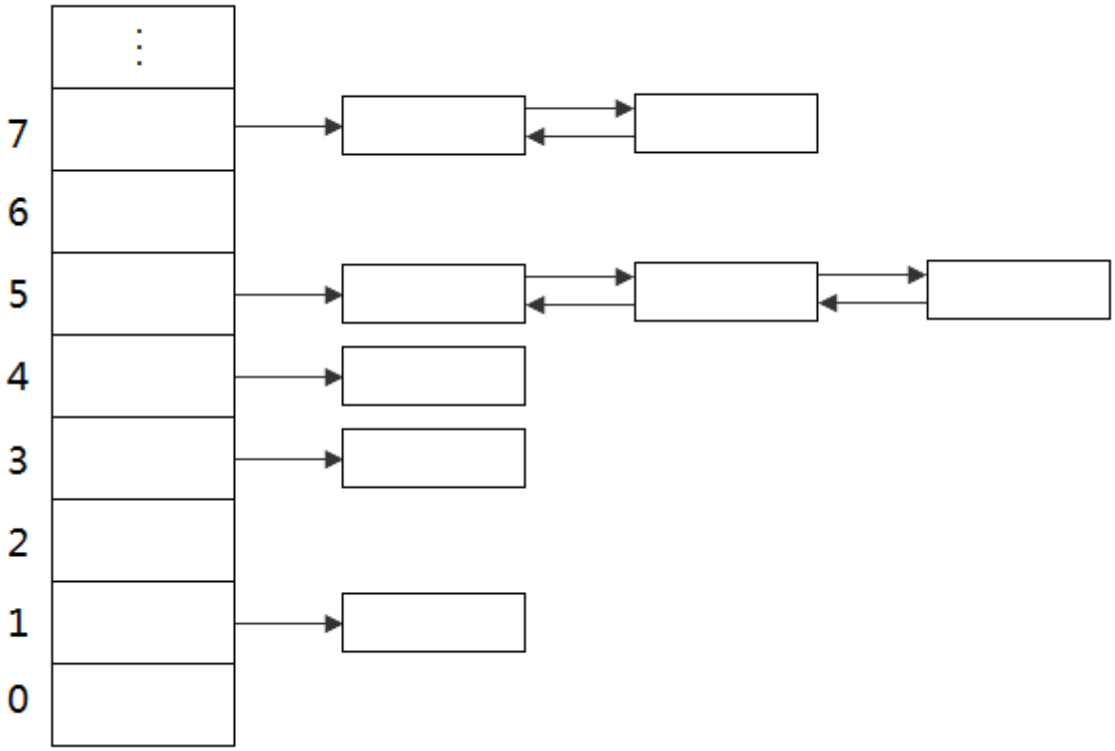
如果在在进行数据传输，那么客户端向服务器发送的 ABOR 命令是通过紧急模式来传输的，否则是按正常模式传输的。所以要处理 ABOR 命令，需要开启紧急模式接收数据。

服务器接收这个命令时可能处在两种状态：(1)FTP 服务命令已经完成，或者(2)FTP 服务命令还在执行中。

第一种情况，服务器关闭数据连接（如果数据连接是打开的）回应 226 代码，表示放弃命令已经成功处理。

第二种情况，服务器放弃正在进行的 FTP 服务，关闭数据连接，返回 426 响应代码，表示请求服务请求异常终止。然后服务器发送 226 响应代码，表示放弃命令成功处理。

3.10 哈希表设计



typedef struct hash hash_t; typedef unsigned int (*hashfunc_t)(unsigned int, void*);
typedef struct hash_node { void *key; void *value; struct hash_node *prev; struct hash_node *next; } hash_node_t;

```
struct hash
{
    unsigned int buckets;
    hashfunc_t hash_func;
    hash_node_t **nodes;
};
```

函数	说明
hash_t* hash_alloc(unsigned int buckets, hashfunc_t hash_func)	创建哈希表
void* hash_lookup_entry(hash_t *hash, void* key, unsigned int key_size)	在哈希表中查找
void hash_add_entry(hash_t *hash, void *key, unsigned int key_size, void *value, unsigned int value_size)	往哈希表中添加一项
void hash_free_entry(hash_t *hash, void *key, unsigned int key_size)	从哈希表中删除一项
hash_node_t** hash_get_bucket(hash_t *hash, void *key)	获取桶地址
hash_node_t* hash_get_node_by_key(hash_t *hash, void *key, unsigned int key_size)	根据 key 获取哈希表中的一个节点

3.11 连接数限制实现

最大连接数限制

每 IP 连接数限制

3.11.1最大连接数限制

将当前连接数保存于变量 num_clients 变量中，然后与配置项 tunable_max_clients 进行比较，如果超过了就不让登录。当一个客户登录的时候 num_clients 加 1，当一个客户端退出的时候，num_clients 减 1。

3.11.2每 IP 连接数限制

维护护两个哈希表
static hash_t *s_ip_count_hash;
static hash_t *s_pid_ip_hash;

将当前 ip 的连接数保存在变量 num_this_ip 中，然后与配置项 tunable_max_per_ip 进行比较，如果超过了就不让登录。当一个客户登录的时候，要在 s_ip_count_hash 更新这个表中的对应表项，即该 ip 对应的连接数要加 1，如果这个表项还不存在，要在表中添加一条记录，并且将 ip 对应的连接数置 1。当一个客户端退出的时候，那么该客户端对应 ip 的连接数要减 1，处理过程是这样的，首先是客户端退出的时候，父进程需要知道这个客户端的 ip，这可以通过在 s_pid_ip_hash 查找得到，得到了 ip 进而我们就可以在 s_ip_count_hash 表中找到对应的连接数，进而进行减 1 操作。

3.12 HELP 命令、STAT 命令、SITE 命令

SITE CHMOD <perm> <file>
SITE UMASK [umask]
SITE HELP