Nginx 邮件代理研究

目录

1 Nginx 邮件代理服务器功能

Nginx 邮件代理服务器，它不会提供实际的邮件服务器功能，而是把客户端的请求代理到上游的邮件服务器中。Nginx 并不是简单地透传邮件协议到上游，它还有一个认证的过程，如图 1-1 所示。

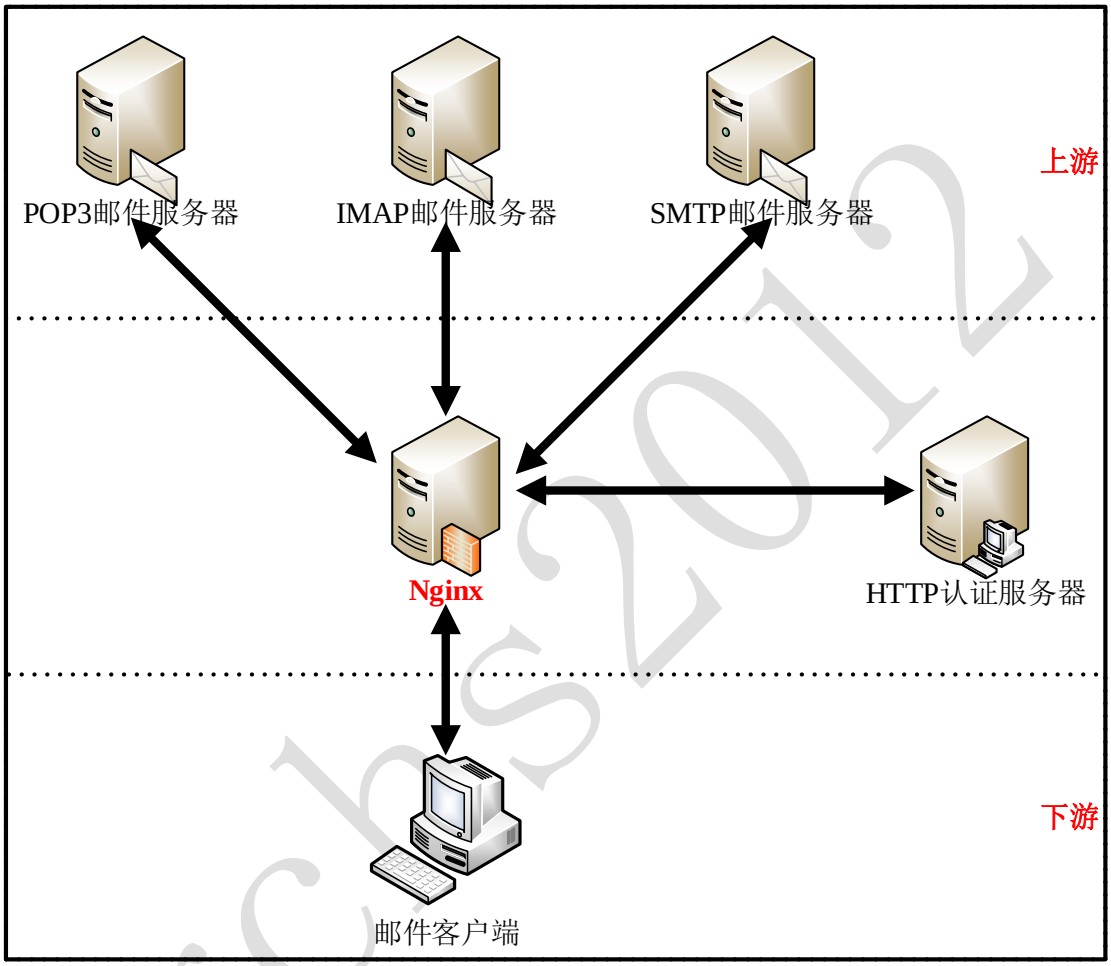

图 1-1 Nginx 在邮件代理场景中的位置

从图 1 中可以看出，Nginx 在与下游客户端交互过程中，还会访问认证服务器，只有认证服务器通过了并且被告知 Nginx 上游的邮件服务器地址后，Nginx 才会向上游的邮件服务器发起通信请求。同时，Nginx 可以解析客户端的协议获得必要的信息，接下来它还可以根据客户端发来的信息快速、独立地与邮件服务器做简单的认证交互，之后才会开始在上、下游之间透传 TCP 流。这些行为都意味着 Nginx 的高并发特性将会降低上游邮件服务器的并发压力。

Nginx 与下游客户端、上游邮件服务器间都是使用邮件协议，而与认证服务器之间却是通过类似 HTTP 的形式进行通信的。例如，发往认证服务器的请求如下所示：

```
Get auth.php HTTP1.0
Host: 192.190.20.162
Auth-Method: plain
Auth-User: nginxtest@MyTest.com
```

```
Auth-Pass: 123456
Auth-Protocol: smtp
Auth-Login-Attempt: 1
Client-IP: 192.190.20.103
```

而认证服务器会返回最常见的成功响应，即类似下面的字符流：

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 192.190.20.200
Auth-Port: 25
```

一般情况下，客户端发起邮件请求在经过 Nginx 这个邮件代理服务器后，网络通信过程如图 1-2 所示。

从网络通信的角度来看，Nginx 实现邮件代理功能时会把一个请求分为以下 4 个阶段。

- 接收并解析客户端初始请求的阶段。
- 向认证服务器验证请求合法性，并获取上游邮件服务器地址的阶段。
- Nginx 根据用户信息多次与上游邮件服务器交互验证合法性的阶段。
- Nginx 在客户端与上游邮件服务器间纯粹透传 TCP 流的阶段。

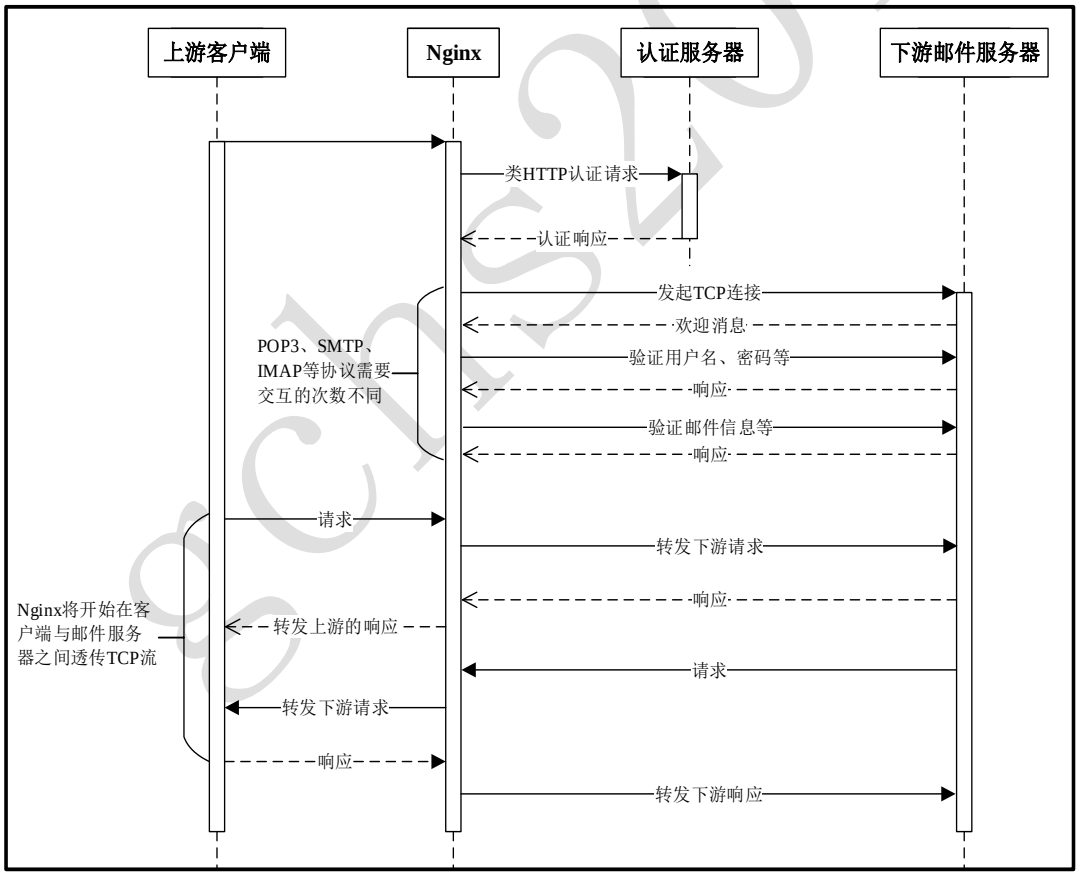

图 1-2 邮件代理功能的示意序列图

2 邮件模块的处理框架

2.1 一个请求的 8 个独立处理阶段

对于 Nginx 邮件框架而言，通常可以把请求的处理过程分为 8 个阶段。由于 Nginx 是异步的、非阻塞的处理方式，所有负责独立功能的一个（或者几个）方法可能被 epoll 或者定时器无数次地驱动、调度，故而可以把相同代码可能被反复多次调用的过程称为一个阶段。下面按照这种划分方式，把请求分为 8 个阶段，如图 2-1 所示。

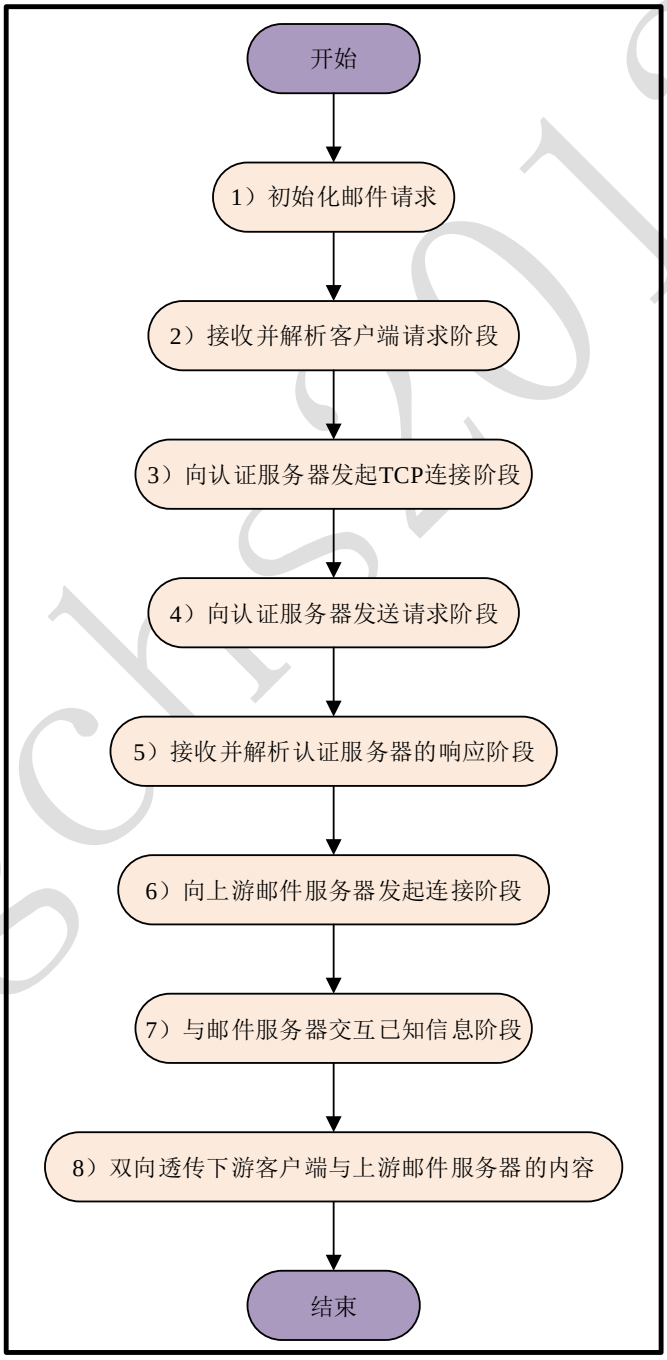

图 2-1 邮件框架中处理一个请求的主要阶段

2.2 邮件类模块定义

邮件模块是 NGX_CORE_MODULE 类型的 ngx_mail_module 模块，它定义了一种新的模块类型，叫做 NGX_MAIL_MODULE。同时，它会管理所有 NGX_MAIL_MODULE 类型的邮件模块。这些邮件模块的 ctx 成员指向的抽象接口叫做 ngx_mail_module_t，如下所示：

```
static ngx_mail_module_t    ngx_mail_core_module_ctx = {
    NULL,                                   /* protocol */
    ngx_mail_core_create_main_conf,         /* create main configuration */
    NULL,                                   /* init main configuration */
    ngx_mail_core_create_srv_conf,          /* create server configuration */
    ngx_mail_core_merge_srv_conf            /* merge server configuration */
};

ngx_module_t    ngx_mail_core_module = {
    NGX_MODULE_V1,
    &ngx_mail_core_module_ctx,              /* module context */
    ngx_mail_core_commands,                 /* module directives */
    NGX_MAIL_MODULE,                        /* module type */
    NULL,                                   /* init master */
    NULL,                                   /* init module */
    NULL,                                   /* init process */
    NULL,                                   /* init thread */
    NULL,                                   /* exit thread */
    NULL,                                   /* exit process */
    NULL,                                   /* exit master */
    NGX_MODULE_V1_PADDING
};
```

注：此结构在 ngx_mail_core_module.c 文件中。

```
typedef struct {
    /* POP3、SMTP、IMAP 邮件模块提取出的通用接口 */
    ngx_mail_protocol_t          *protocol;

    /* 创建用于存储 main 级别配置项的结构体，该结构体中的成员将保存直属于 mail{} 块的配置项参数 */
    void                         *(*create_main_conf)(ngx_conf_t *cf);

    /* 解析完 main 级别配置项后被回调 */
    char                         *(*init_main_conf)(ngx_conf_t *cf, void *conf);

    /* 创建用于存储 srv 级别配置项的结构体，该结构体中的成员将保存直属于 server{} 块的配置项参数 */
    void                         *(*create_srv_conf)(ngx_conf_t *cf);
```

```
    /* svr 级别可能存在与 main 级别同名的配置项，该回调方法会给具体的邮件模块提供一个手段，*/
    /* 以便从 prev 和 conf 参数中获取到已经解析完毕的 main 和 srv 配置项结构体，自由地重新修改它们的值 */
    char                                *(*merge_srv_conf)(ngx_conf_t *cf, void *prev,
                                            void *conf);
} ngx_mail_module_t;
```

注：此结构在 ngx_mail.h 文件中。

每一个邮件模块都会实现 ngx_mail_module_t 接口。除了最上面的 protocol 成员以外，其实 ngx_mail_module_t 与 ngx_http_module_t 非常相似，当然，那些同名成员的功能也是相似的。下面看一下这个 protocol 接口定义了哪些内容。

```
/* 4 个 POP3、SMTP、IMAP 等应用级别的邮件模块所要实现的接口方法 */
typedef void (*ngx_mail_init_session_pt)(ngx_mail_session_t *s,
    ngx_connection_t *c);
typedef void (*ngx_mail_init_protocol_pt)(ngx_event_t *rev);
typedef void (*ngx_mail_auth_state_pt)(ngx_event_t *rev);
typedef ngx_int_t (*ngx_mail_parse_command_pt)(ngx_mail_session_t *s);


/* 定义了 POP3、SMTP、IMAP 等应用级别的邮件模块加入到邮件 */
/* 框架时所要实现的接口以及需要遵循的规则 */
struct ngx_mail_protocol_s {
    /* 模块名称 */
    ngx_str_t                       name;

    /* 当前邮件模块中所要监听的最常用的 4 个端口 */
    in_port_t                       port[4];

    /* 邮件类型，目前 type 值仅可取值为: NGX_MAIL_POP3_PROTOCOL、*/
    /* NGX_MAIL_IMAP_PROTOCOL、NGX_MAIL_SMTP_PROTOCOL */
    ngx_uint_t                      type;

    /* 与客户端建立起 TCP 连接后的初始化方法 */
    ngx_mail_init_session_pt        init_session;

    /* 接收、解析客户端请求的方法 */
    ngx_mail_init_protocol_pt       init_protocol;

    /* 解析客户端邮件协议的接口方法，由 POP3、SMTP、IMAP 等模块实现 */
    ngx_mail_parse_command_pt       parse_command;

    /* 认证客户端请求的方法 */
    ngx_mail_auth_state_pt          auth_state;
```

```
    /* 当处理过程中出现没有预见到的错误时，将会返回 */
    /* internal_server_error 指定的响应到客户端 */
    ngx_str_t                       internal_server_error;
};
```

注：此结构在 ngx_mail.h 文件中。

可以看到，ngx_mail_protocol_t 接口定义了 POP3、SMTP、IMAP 等应用级别的邮件模块加入到邮件框架时所要实现的接口以及需要遵循的规则。

2.3 邮件框架的初始化

当 nginx.conf 文件中出现 mail{}或者 imap{}配置项时，ngx_mail_module 模块就从 **ngx_mail_block** 方法开始它的初始化过程（与 HTTP 框架非常相似），如图 2-2 所示。其中最后一步中设置的 TCP 连接建立成功后的回调方法为 **ngx_mail_init_connection**。

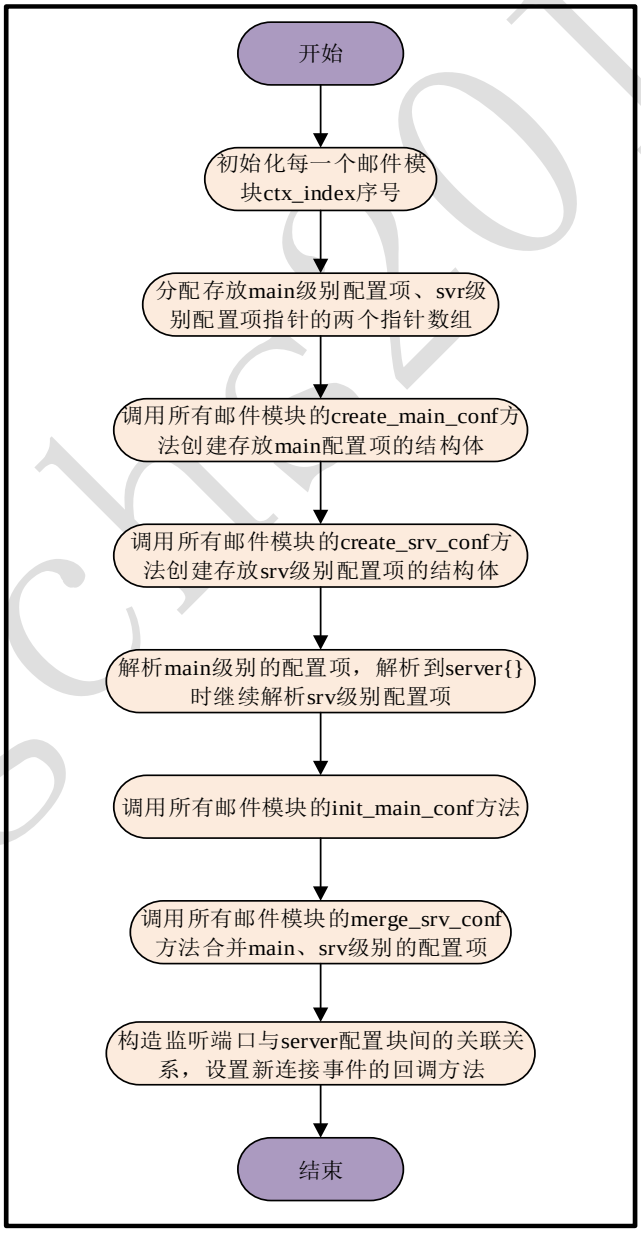

图 2-2 邮件框架初始化的流程图

3 初始化请求

Nginx 与客户端建立 TCP 连接后，将会回调 ngx_mail_init_connection 方法开始初始化邮件协议，这是在处理每个邮件请求前必须要做的工作。其中，初始化请求时将会创建核心结构体：ngx_mail_session_t。

3.1 核心结构体（ngx_mail_session_t）

```
typedef struct {
    /* 与上游邮件服务器间的连接 */
    ngx_peer_connection_t    upstream;

    /* 用于缓存上、下游间 TCP 消息的内存缓冲区，内存大小 */
    /* 由 nginx.conf 文件中的 proxy_buffer 配置项决定 */
    ngx_buf_t                *buffer;
} ngx_mail_proxy_ctx_t;


/* 该结构体保存了一个邮件请求的生命周期里所有可能用到的元素 */
typedef struct {
    uint32_t                 signature;          /* "MAIL" */

    /* 下游客户端与 nginx 之间的连接 */
    ngx_connection_t         *connection;

    /* out 中可以存放需要向下游客户端发送的内容 */
    ngx_str_t                out;

    /* 这个缓冲区用于接收来自客户端的请求。这个缓冲区中所使用的 */
    /* 内存大小与请求是有关系的，对于 POP3 请求固定为 128 字节，对于 */
    /* SMTP 请求，由 nginx.conf 配置文件中的 smtp_client_buffer 配置项 */
    /* 决定，对于 IMAP 请求，则由 imap_client_buffer 配置项决定 */
    ngx_buf_t                *buffer;

    /* ctx 将指向一个指针数组，它的含义与 HTTP 请求的 ngx_http_request_t */
    /* 结构体中的 ctx 一致，保存着这个请求中各个邮件模块的上下文结构体指针 */
    void                     **ctx;

    /* main 级别配置结构体组成的指针数组 */
    void                     **main_conf;

    /* srv 级别配置结构体组成的指针数组 */
```

```
void                        **srv_conf;

ngx_resolver_ctx_t        *resolver_ctx;

/* 请求经过认证后 nginx 就开始代理客户端与邮件服务器间的通信了，*/
/* 这时会生成 proxy 上下文用于此目的 */
ngx_mail_proxy_ctx_t     *proxy;

/* 表示与邮件服务器交互时，当前处理哪种状态。*/
/* 对于 POP3 请求来说，会隶属于 ngx_pop3_state_e 定义的 7 种状态；*/
/* 对于 IMAP 请求来说，会隶属于 ngx_imap_state_e 定义的 8 种状态；*/
/* 对于 SMTP 请求来说，会隶属于 ngx_smtp_state_e 定义的 13 种状态 */
ngx_uint_t                  mail_state;

/* 邮件协议类型目前仅有以下 3 个 */
/* #define NGX_MAIL_POP3_PROTOCOL 0 */
/* #define NGX_MAIL_IMAP_PROTOCOL 1    */
/* #define NGX_MAIL_SMTP_PROTOCOL 2 */
unsigned                    protocol:3;

/* blocked 为 1 时表示当前的读或写操作需要被阻塞 */
unsigned                    blocked:1;

/* quit 为 1 时表示请求需要结束 */
unsigned                    quit:1;

/* 以下 3 个标志位仅在解析具体的邮件协议时由邮件框架使用 */
unsigned                    quoted:1;
unsigned                    backslash:1;
unsigned                    no_sync_literal:1;

/* 当使用 SSL 协议时，该标志位为 1 说明使用 TLS 传输层安全协议 */
unsigned                    starttls:1;
unsigned                    esmtp:1;

/* 表示与认证服务器交互时的记录认证方式。目前有 6 个预设值，分别是：*/
/* #define NGX_MAIL_AUTH_PLAIN              0 */
/* #define NGX_MAIL_AUTH_LOGIN              1 */
/* #define NGX_MAIL_AUTH_LOGIN_USERNAME 2 */
/* #define NGX_MAIL_AUTH_APOP               3 */
/* #define NGX_MAIL_AUTH_CRAM_MD5           4 */
/* #define NGX_MAIL_AUTH_NONE               5 */
unsigned                    auth_method:3;
```

```
/* 用于认证服务器的标志位，为 1 时表示得知认证服务器要求暂缓接收响应，*/
/* 这时 nginx 会继续等待认证服务器的后续响应 */
unsigned                auth_wait:1;

/* 用于验证的用户名，在与认证服务器交互后会被设为认证服务器返回的 */
/* 响应中的 Auth-User 头部 */
ngx_str_t               login;


/* 相对于 login 用户名的密码，在与认证服务器交互后会被设为认证服务器 */
/* 返回的响应中的 Auth-Pass 头部 */
ngx_str_t               passwd;

/* 作为 Auth-Salt 验证的信息 */
ngx_str_t               salt;

/* 以下 3 个成员仅用于 IMAP 通信 */
ngx_str_t               tag;
ngx_str_t               tagged_line;
ngx_str_t               text;

/* 当前连接上对应的 nginx 服务器地址 */
ngx_str_t              *addr_text;

/* 主机地址 */
ngx_str_t               host;

/* 以下 3 个成员仅用于 SMTP 的通信 */
ngx_str_t               smtp_helo;
ngx_str_t               smtp_from;
ngx_str_t               smtp_to;

ngx_str_t               cmd;

/* 在与邮件服务器交互时（即与认证服务器交互之后，透传上下游 */
/* TCP 流之前），command 表示解析自邮件服务器的消息类型 */
ngx_uint_t              command;

/* args 动态数组中会存放来自下游客户端的邮件协议中的参数 */
ngx_array_t             args;

/* 当前请求尝试访问认证服务器验证的次数 */
ngx_uint_t              login_attempt;
```

```
    /* used to parse POP3/IMAP/SMTP command */
    /*  以下成员用于解析 POP3/IMAP/SMTP 等协议的命令行  */
    ngx_uint_t                 state;
    u_char                     *cmd_start;
    u_char                     *arg_start;
    u_char                     *arg_end;
    ngx_uint_t                 literal_len;
} ngx_mail_session_t;
```

注：此结构在 ngx_mail.h 文件中。

3.2 初始化邮件请求流程

初始化邮件流程，如图 3-1 所示。

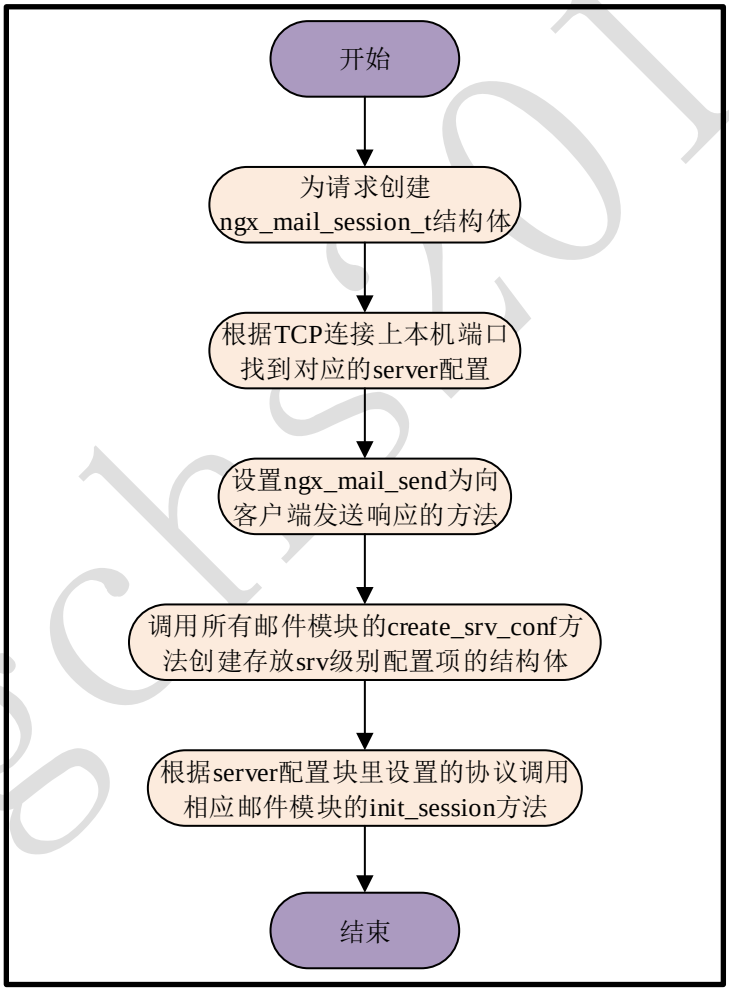

图 3-1 初始化邮件请求的流程

实际上，初始化流程中最关键的一步就是调用 POP3、SMTP、IMAP 等具体邮件模块实现 ngx_mail_protocol_t 接口中的 init_session 方法，这些邮件模块会根据自己处理的协议类型初始化 ngx_mail_session_t 结构体。在 POP3、SMTP、IMAP 邮件模块内实现的 init_session 方法中，都会设置由各自实现的 init_protocol 方法接收、解析客户端请求。

3.3 接收并解析客户端请求

无论是 POP3、SMTP 还是 IMAP 邮件模块，在处理客户端的请求时，都是使用 ngx_mail_protocol_t 接口中的 init_protocol 方法完成的，它们的流程十分相似：首先反复地接收客户端请求，并使用状态机解析是否收到足够的信息，直到接收了完整的信息后才会跳到下一个邮件认证阶段执行（通过调用 ngx_mail_auth 方法）。

使用状态机解析来自客户端的 TCP 流的方法其实就是通过 ngx_mail_protocol_t 接口中的 parse_command 方法来完成的，POP3、SMTP、IMAP 邮件模块实现的 parse_command 方法都在 ngx_mail_parser.c 源文件中。

4 邮件认证

邮件认证工作由 ngx_mail_auth 方法执行。邮件认证服务器的地址在 nginx.conf 文件的 auth_http 配置项中设置（参见 13.1 节），这一认证流程相对独立，其认证功能是由 ngx_mail_auth_http_module 邮件模块提供的。在与认证邮件服务器打交道的过程中，结构体 ngx_mail_auth_http_ctx_t 会贯穿其始终，它保存有连接、请求内容、响应内容、解析状态等必要的成员，在认证完邮件后将会通过销毁内存池来销毁这个结构体。

4.1 核心结构体（ngx_mail_auth_http_ctx_t）

```
typedef struct ngx_mail_auth_http_ctx_s    ngx_mail_auth_http_ctx_t;

/*  解析认证服务器 HTTP 响应的方法  */
typedef void (*ngx_mail_auth_http_handler_pt)(ngx_mail_session_t *s,
      ngx_mail_auth_http_ctx_t *ctx);

struct ngx_mail_auth_http_ctx_s {
      /* request 缓冲区保存着发往认证服务器的请求。它是根据解析客户端请求得到的  */
      /* ngx_mail_session_t，使用 ngx_mail_auth_http_create_request 方法构造出的  */
      /*  内存缓冲区。这里的请求是一种类 HTTP 的请求  */
      ngx_buf_t                        *request;

      /*  保存认证服务器返回的类 HTTP 响应的缓冲区。缓冲区指向的内存大小固定为 1K
*/
      ngx_buf_t                        *response;

      /* nginx 与认证服务器间的连接  */
      ngx_peer_connection_t            peer;

      /*  解析来自认证服务器类 HTTP 的响应行、头部的方法  */
      /*  默认为 ngx_mail_auth_http_ignore_status_line 方法  */
```

```
    ngx_mail_auth_http_handler_pt      handler;

    /* 在使用状态机解析认证服务器返回的类 HTTP 响应时，使用 state 表示解析状态 */
    ngx_uint_t                         state;

    /* ngx_mail_auth_http_parse_header_line 方法负责解析认证服务器发来的响应中类 */
    /* HTTP 的头部，以下 4 个成员用于解析响应头部 */
    u_char                             *header_name_start;
    u_char                             *header_name_end;
    u_char                             *header_start;
    u_char                             *header_end;

    /* 认证服务器返回的 Auth-Server 头部 */
    ngx_str_t                          addr;

    /* 认证服务器返回的 Auth-Port 头部 */
    ngx_str_t                          port;

    /* 错误信息 */
    ngx_str_t                          err;

    /* 错误信息构成的字符串 */
    ngx_str_t                          errmsg;

    /* 错误码构成的字符串。如果认证服务器返回的头部里有 Auth-Error-Code，那么将会 */
    /* 设置到 errcode 中。errmsg 和 errcode 在发生错误时会直接将其作为响应发给客户端 */
    ngx_str_t                          errcode;

    /* 认证服务器返回 Auth-Wait 头部时带的时间戳将会被设到 sleep 成员中，而 Nginx 等待的 */
    /* 时间也将由 sleep 维护，当 sleep 降为 0 时将会设置 quit 标志位为 1，表示请求非正常结束，*/
    /* 把错误码返回给用户 */
    time_t                             sleep;

    /* 用于邮件认证的独立内存池，它的初始大小为 2KB */
    ngx_pool_t                         *pool;
};
```

注：此结构在 ngx_mail_auth_http_module.c 文件中。

4.2 与认证服务器建立连接

ngx_mail_auth 方法所做的工作，包括初始化与认证服务器交互之前的工作、发起 TCP 连接等。Nginx 与下游客户端间 TCP 连接上的读事件处理方法为 ngx_mail_auth_http_block_read，这个方法所做的唯一工作其实就是再次调用 ngx_handle_read_event 方法把读事件又添加到 epoll 中，这意味着它不会读取任何客户端发来的请求，但同时保持着读事件被 epoll 监控。在与认证服务器间 TCP 连接上，写事件的处理方法为 ngx_mail_auth_http_write_handler，它负责把构造出的 request 缓冲区中的请求发送给认证服务器；读事件的处理方法为 ngx_mail_auth_http_read_handler，这个方法在接收到认证服务器的响应后会调用 ngx_mail_auth_http_ignore_status_line 方法首先解析 HTTP 响应行。

4.3 接收并解析响应

接收并解析认证服务器响应的方法是 ngx_mail_auth_http_read_handler，该方法要同时负责解析响应行和 HTTP 头部。当没有收到足够的 TCP 流供状态机解析时，都会期待 epoll 下一次重新调度。在全部解析完响应后，将可以得知认证是否通过，如果请求合法，那么可以从 HTTP 响应头部中得到上游邮件服务器的地址，接着通过调用 ngx_mail_proxy_init 方法进入与邮件服务器交互的阶段。

5 与上游邮件服务器间的认证交互

对于 POP3、SMTP、IMAP 来说，客户端与邮件服务器之间最初的交互目的都不太相同。例如，对于 POP3 和 IMAP 来说，与邮件服务器间的 TCP 连接一旦建立成功，邮件服务器会发送一个欢迎信息，接着客户端（此时，Nginx 是邮件服务器的客户端）发送用户名，在邮件服务器返回成功后再发送密码，等邮件服务器验证通过后，才会进入到邮件处理阶段：对于 Nginx 这个邮件代理服务器来说，就是进入到纯粹地透传 Nginx 与上、下游间两个 TCP 连接之间的数据流。但对于 SMTP 来说，这个交互过程又有所不同，进入邮件处理阶段前需要交互传输邮件来源地址、邮件目标地址（也就是 From…To…）等信息。

无论如何，Nginx 作为邮件代理服务器在接收到客户端的请求，并且收集到足够进行认证的信息后，将会由 Nginx 与上游的邮件服务器进行独立的交互，直到邮件服务器认为可以进入到处理阶段时，才会开始透传协议。这一阶段将围绕着 ngx_mail_proxy_ctx_t 结构体中的成员进行。下面以 POP3 协议为例简单地说明 Nginx 是如何与邮件服务器交互的。

5.1 核心结构体（ngx_mail_proxy_ctx_t）

ngx_mail_session_t 结构体中的 proxy 成员指向 ngx_mail_proxy_ctx_t 结构体，该结构体含有 Nginx 与上游间的连接 upstream，以及与上游通信时接收上游 TCP 消息的缓冲区。proxy 成员最初也是 NULL 空指针，直到调用 ngx_mail_proxy_init 方法后才会为 proxy 指针分配内存。

```
typedef struct {
```

```
    /* 与上游邮件服务器间的连接 */
    ngx_peer_connection_t     upstream;

    /* 用于缓存上、下游间 TCP 消息的内存缓冲区，内存大小 */
    /* 由 nginx.conf 文件中的 proxy_buffer 配置项决定 */
    ngx_buf_t                 *buffer;

    /* BEGIN: Added by zhangcan, 2018/6/8 */
#if (NGX_MAIL_SSL)
    ngx_ssl_t                 *ssl;
#endif
    /* END:    Added by zhangcan, 2018/6/8 */
} ngx_mail_proxy_ctx_t;
```

5.2 与上游邮件服务器发起连接

根据 ngx_mail_proxy_init 方法可以启动 Nginx 与上游邮件服务器间的交互（具体可以查看该接口，里面已经添加关键注释），可以看到，其中最重要的工作在于分配了 ngx_mail_proxy_ctx_t 结构体，并为成员 buffer 分配了内存缓冲区，用于接收上游的 TCP 消息，同时使用 upstream 与上游建立了 TCP 连接，最后针对不同的邮件协议分别设置了 ngx_mail_proxy_pop3_handler 、 ngx_mail_proxy_imap_handler 或者 ngx_mail_proxy_smtp_handler 方法，用于 Nginx 与上游邮件服务器间的交互。

5.3 与上游邮件服务器认证交互过程

由于每种协议的交互过程都不相同，可以参照 POP3 协议为例，根据 ngx_mail_proxy_pop3_handler 方法查看与上游邮件服务器认证交互过程（具体可以查看该接口，里面已经添加关键注释）。一旦收到用户名、密码验证通过的消息，就会由 ngx_mail_proxy_handler 方法进入透传上、下游 TCP 流的阶段。

6 透传上游邮件服务器与下游客户端间的流

ngx_mail_proxy_handler 方法同时负责处理上、下游间的四个事件（两个读事件、两个写事件）。该方法将完全实现上、下游邮件协议之间的透传，本节将通过直接研究这个方法来看看如何用固定大小的缓存实现透传功能（有些类似于 upstream 机制转发响应时仅用了固定缓存的模式，但 upstream 机制只是单向的转发，而透传则是双向的转发）。下面先来介绍双向转发 TCP 流时将会用到的两个缓冲区：ngx_mail_session_t 中的 buffer 缓冲区用于转发下游客户端的消息给上游的邮件服务器，而 ngx_mail_proxy_ctx_t 中的 buffer 缓冲区则用于转发上游邮件服务器的消息给下游的客户端。在这两个 ngx_buf_t 类型的缓冲区中，pos 指针指向待转发消息的起始地址，而 last 指针指向最后一次接收到的消息的末尾。当 pos 等于 last 时，意味着全部缓存消息都转发完了，这时会把 pos 和 last 都指向缓冲区的首部 start 指

针，相当于清空缓冲区以便再次复用完整的缓冲区。具体可以查看该接口，已经添加关键注释。

7 配置文件

7.1 配置方式

邮件代理配置方式，有以下四种：

- 下游与 Nginx 普通连接，Nginx 与上游普通连接
- 下游与 Nginx 普通连接，Nginx 与上游 SSL 连接
- 下游与 Nginx SSL 连接，Nginx 与上游普通连接
- 下游与 Nginx SSL 连接，Nginx 与上游 SSL 连接

7.1.1 下游与 Nginx 普通连接，Nginx 与上游普通连接

```
mail {
    # 邮件认证服务器的访问 URL
    auth_http             192.190.20.162:80/mail_auth.php;

    # 认证超时时长 10 秒
    auth_http_timeout   10;

    # SMTP Auth Proxy
    server {
        listen                  25;
        protocol                smtp;
        proxy                   on;
        proxy_buffer            4k;
        xclient                 off;
        smtp_auth               plain login;
        smtp_capabilities   "SIZE 20480000"   8BITMIME;

        # 设置接收初始客户端请求的缓冲区大小为 4k
        smtp_client_buffer 4k;
    }

    # POP3 Auth Proxy
    server {
        # 对于 POP3 协议，通常监听 110 端口，POP3 协议接收客户端请求缓冲区
        # 固定为 128 字节，配置文件中无法设置
        listen                  110;
```

```
        protocol        pop3;
        proxy           on;
        pop3_auth       plain apop cram-md5;
        pop3_capabilities   "TOP" "USER" "UIDL";
    }

    # IMAP Auth Proxy
    server {
        listen          143;
        protocol        imap;
        proxy           on;
        imap_auth       plain login cram-md5;
        imap_capabilities   "IMAP4" "IMAP4rev1" "UIDPLUS";

        # 设置接收初始客户端请求的缓冲区大小为 4k
        imap_client_buffer  4k;
    }
}
```

7.1.2 下游与 Nginx 普通连接，Nginx 与上游 SSL 连接

```
mail {
    # 邮件认证服务器的访问 URL
    auth_http           192.190.20.162:80/mail_auth.php;

    # 认证超时时长 10 秒
    auth_http_timeout   10;

    # SMTP Auth Proxy
    server {
        listen          25;
        protocol        smtp;
        proxy           on;
        proxy_buffer    4k;
        xclient         off;
        smtp_auth       plain login;
        smtp_capabilities   "SIZE 20480000"   8BITMIME;

        # 设置接收初始客户端请求的缓冲区大小为 4k
        smtp_client_buffer 4k;

        mail_ssl            on;
        mail_ssl_protocols  SSLv2 SSLv3 TLSv1 TLSv1.1 TLSv1.2;
```

```
        mail_ssl_ciphers
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
    }
}
```

7.1.3 下游与 Nginx SSL 连接，Nginx 与上游普通连接

```
mail {
    # 邮件认证服务器的访问 URL
    auth_http             192.190.20.162:80/mail_auth.php;

    # 认证超时时长 10 秒
    auth_http_timeout    10;

    starttls                    on;
    ssl_certificate             mail/server.crt;
    ssl_certificate_key         mail/server.key;
    ssl_protocols               SSLv2 SSLv3 TLSv1 TLSv1.1 TLSv1.2;
    ssl_session_cache           shared:SSL:10m;
    ssl_session_timeout         5m;
    ssl_prefer_server_ciphers   on;
    ssl_ciphers
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

    # SMTP Auth Proxy
    server {
        listen                  25;
        protocol                smtp;
        proxy                   on;
        proxy_buffer            4k;
        xclient                 off;
        smtp_auth               plain login;
        smtp_capabilities       "SIZE 20480000"    8BITMIME;

        # 设置接收初始客户端请求的缓冲区大小为 4k
        smtp_client_buffer 4k;
    }
}
```

7.1.4 下游与 Nginx SSL 连接，Nginx 与上游 SSL 连接

```
mail {
```

```
    # 邮件认证服务器的访问 URL
    auth_http             192.190.20.162:80/mail_auth.php;

    # 认证超时时长 10 秒
    auth_http_timeout    10;

    starttls                    on;
    ssl_certificate             mail/server.crt;
    ssl_certificate_key         mail/server.key;
    ssl_protocols               SSLv2 SSLv3 TLSv1 TLSv1.1 TLSv1.2;
    ssl_session_cache           shared:SSL:10m;
    ssl_session_timeout         5m;
    ssl_prefer_server_ciphers   on;
    ssl_ciphers
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

    # SMTP Auth Proxy
    server {
        listen                  25;
        protocol                smtp;
        proxy                   on;
        proxy_buffer            4k;
        xclient                 off;
        smtp_auth               plain login;
        smtp_capabilities       "SIZE 20480000"    8BITMIME;

        # 设置接收初始客户端请求的缓冲区大小为 4k
        smtp_client_buffer 4k;

        mail_ssl                on;
        mail_ssl_protocols      SSLv2 SSLv3 TLSv1 TLSv1.1 TLSv1.2;
        mail_ssl_ciphers
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
    }
}
```

7.2 认证脚本

该 PHP 脚本主要作用是通过 nginx 发送的请求数据，获取上游邮件服务器的 IP 和端口，并将上游邮件服务器 IP 和端口返回给 nginx。

```
<?php
/**
```

```
*  主要通过向邮件服务器进行认证
*
* @author       zhangcan
* @version      v1.0
*/

if (!isset($_SERVER["HTTP_AUTH_USER"]) || !isset($_SERVER["HTTP_AUTH_PASS"])) {
        response_failed(0);
}

$username = $_SERVER["HTTP_AUTH_USER"];
$userpass = $_SERVER["HTTP_AUTH_PASS"];
$protocal = $_SERVER["HTTP_AUTH_PROTOCOL"];
$backport = 110;

// 通过协议判断端口
if ($protocal == "imap") {
        $backport = 143;
} elseif ($protocal == "smtp") {
        $backport = 9999;
}

// 进行用户认证
$auth = user_auth($username, $userpass);
if ($auth) {
        response_success("192.190.20.200", $backport);

        // response_success("192.180.10.208", $backport);
} else {
        response_failed(-2);
}

// 自定义认证，此接口自己进行实现，目前只是简单返回 true
function user_auth($user, $pass) {
        return true;
}

// 错误处理函数
function response_failed($code) {
        switch ($code) {
                case 0:
                        header("Auth-Status: Parmeter lost");
                        break;
                case -1:
```

```
                header("Auth-Status: No Back-end Server");
                break;
            case -2:
                header("Auth-Status: Invalid login or password");
                break;
        }
        exit();
}

// 认证通过
function response_success($server, $port) {
        header("Auth-Status: OK");
        header("Auth-Server: $server");
        header("Auth-Port: $port");
        exit();
}
?>
```

8 添加代码部分

8.1 ngx_mail.h

- 在 ngx_mail_core_srv_conf_t 结构中添加 mail_ssl、mail_ssl_protocols、mail_ssl_ciphers 三个命令。

```
typedef struct {
        ngx_mail_protocol_t       *protocol;

        ngx_msec_t                 timeout;
        ngx_msec_t                 resolver_timeout;

        ngx_flag_t                 so_keepalive;

        ngx_str_t                  server_name;

        u_char                    *file_name;
        ngx_int_t                  line;

        ngx_resolver_t            *resolver;

        /* BEGIN: Added by zhangcan, 2018/6/8 */
#if (NGX_MAIL_SSL)
```

```
    ngx_flag_t          mail_ssl;                // nginx 与上游是否建立 SSL 连接开关
    ngx_uint_t          mail_ssl_protocols;   // SSL 协议，例如: SSLv2、TLSv1.1 等
    ngx_str_t           mail_ssl_ciphers;      // SSL 加密方式，例如: RC4+RSA、+SSLv2 等
#endif
    /* END:     Added by zhangcan, 2018/6/8 */

    /* server ctx */
    ngx_mail_conf_ctx_t       *ctx;
} ngx_mail_core_srv_conf_t;
```

- 在 ngx_mail_proxy_ctx_t 结构中添加 ssl 字段，用于保存 nginx 与上游服务器之间的 SSL 连接信息。

```
typedef struct {
    /*  与上游邮件服务器间的连接  */
    ngx_peer_connection_t     upstream;

    /*  用于缓存上、下游间 TCP 消息的内存缓冲区，内存大小  */
    /*  由 nginx.conf 文件中的 proxy_buffer 配置项决定  */
    ngx_buf_t                       *buffer;

/* BEGIN: Added by zhangcan, 2018/6/8 */
#if (NGX_MAIL_SSL)
    ngx_ssl_t                  *ssl;
#endif
/* END:     Added by zhangcan, 2018/6/8 */
} ngx_mail_proxy_ctx_t;
```

8.2 ngx_mail_core_module.c

- 定义 SSL 协议类型。

```
/* BEGIN: Added by zhangcan, 2018/6/7 */
#if (NGX_MAIL_SSL)
static ngx_conf_bitmask_t    ngx_mail_proxy_ssl_protocols[] = {
    { ngx_string("SSLv2"), NGX_SSL_SSLv2 },
    { ngx_string("SSLv3"), NGX_SSL_SSLv3 },
    { ngx_string("TLSv1"), NGX_SSL_TLSv1 },
    { ngx_string("TLSv1.1"), NGX_SSL_TLSv1_1 },
    { ngx_string("TLSv1.2"), NGX_SSL_TLSv1_2 },
    { ngx_null_string, 0 }
};
#endif
/* END:     Added by zhangcan, 2018/6/7 */
```

➢ 添加解析 mail_ssl、mail_ssl_protocols、mail_ssl_ciphers 三个命令方法。

```
static ngx_command_t    ngx_mail_core_commands[] = {

    { ngx_string("server"),
      NGX_MAIL_MAIN_CONF|NGX_CONF_BLOCK|NGX_CONF_NOARGS,
      ngx_mail_core_server,
      0,
      0,
      NULL },

    ……

    { ngx_string("resolver_timeout"),
      NGX_MAIL_MAIN_CONF|NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_core_srv_conf_t, resolver_timeout),
      NULL },

    /* BEGIN: Added by zhangcan, 2018/6/7 */
#if (NGX_MAIL_SSL)
    /*  以下三个命令为用户自定义命令  */
    /* mail_ssl                 --- nginx 与上游是否建立 SSL 连接开关  */
    /* mail_ssl_protocols    --- SSL 相关协议，例如: SSLv2、TLSv1.2 等  */
    /* mail_ssl_ciphers       --- SSL 加密方式，例如: RC4+RSA、+SSLv2 等*/
    { ngx_string("mail_ssl"),
      NGX_MAIL_SRV_CONF|NGX_CONF_FLAG,
      ngx_conf_set_flag_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_core_srv_conf_t, mail_ssl),
      NULL },

    { ngx_string("mail_ssl_protocols"),
      NGX_MAIL_SRV_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
      offsetof(ngx_mail_core_srv_conf_t, mail_ssl_protocols),
      &ngx_mail_proxy_ssl_protocols },

    { ngx_string("mail_ssl_ciphers"),
      NGX_MAIL_SRV_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_MAIL_SRV_CONF_OFFSET,
```

```
        offsetof(ngx_mail_core_srv_conf_t, mail_ssl_ciphers),
        NULL },
#endif
    /* END:     Added by zhangcan, 2018/6/7 */

        ngx_null_command
};
```

- 初始化 mail_ssl、mail_ssl_protocols、mail_ssl_ciphers 三个字段。

```
static void *
ngx_mail_core_create_srv_conf(ngx_conf_t *cf)
{
    ngx_mail_core_srv_conf_t    *cscf;

    cscf = ngx_pcalloc(cf->pool, sizeof(ngx_mail_core_srv_conf_t));
    if (cscf == NULL) {
        return NULL;
    }

    /*
     * set by ngx_pcalloc():
     *
     *         cscf->protocol = NULL;
     */

    cscf->timeout = NGX_CONF_UNSET_MSEC;
    cscf->resolver_timeout = NGX_CONF_UNSET_MSEC;
    cscf->so_keepalive = NGX_CONF_UNSET;

    cscf->resolver = NGX_CONF_UNSET_PTR;

    cscf->file_name = cf->conf_file->file.name.data;
    cscf->line = cf->conf_file->line;

    /* BEGIN: Added by zhangcan, 2018/6/8 */
#if (NGX_MAIL_SSL)
    /*  设置初始值  */
    cscf->mail_ssl = NGX_CONF_UNSET;
    cscf->mail_ssl_protocols = 0;
    ngx_str_null(&(cscf->mail_ssl_ciphers));
#endif
    /* END:     Added by zhangcan, 2018/6/8 */

    return cscf;
```

```
}
```

8.3 ngx_mail_proxy_module.c

➢ 在 nginx 与上游服务器建立连接初始化函数中添加是否需要已 SSL 方法进行连接代码。

```
/**************************************************************************
*
      函 数 名 : ngx_mail_proxy_init
      功能描述 : nginx 与上游邮件服务器建立连接初始化过程
      输入参数 : ngx_mail_session_t *s
                 ngx_addr_t *peer
      输出参数 : 无
      返 回 值 : 无
      作      者 : zhangcan
      日      期 : 2018 年 6 月 5 日
**************************************************************************
*/
void
ngx_mail_proxy_init(ngx_mail_session_t *s, ngx_addr_t *peer)
{
      int                                keepalive;
      ngx_int_t                          rc;
      ngx_mail_proxy_ctx_t               *p;
      ngx_mail_proxy_conf_t              *pcf;
      ngx_mail_core_srv_conf_t           *cscf;

      s->connection->log->action = "connecting to upstream";

      cscf = ngx_mail_get_module_srv_conf(s, ngx_mail_core_module);

      if (cscf->so_keepalive) {
            keepalive = 1;

            if (setsockopt(s->connection->fd, SOL_SOCKET, SO_KEEPALIVE,
                           (const void *) &keepalive, sizeof(int))
                == -1)
            {
                  ngx_log_error(NGX_LOG_ALERT, s->connection->log, ngx_socket_errno,
                                "setsockopt(SO_KEEPALIVE) failed");
            }
      }

      /* 创建 ngx_mail_proxy_ctx_t 结构体 */
```

```
    p = ngx_pcalloc(s->connection->pool, sizeof(ngx_mail_proxy_ctx_t));
    if (p == NULL) {
        ngx_mail_session_internal_server_error(s);
        return;
    }

    /* proxy 成员最初是 NULL 空指针，直到调用 ngx_mail_proxy_init */
    /*  方法后才会为 proxy 指针分配内存  */
    s->proxy = p;

    p->upstream.sockaddr = peer->sockaddr;
    p->upstream.socklen = peer->socklen;
    p->upstream.name = &peer->name;
    p->upstream.get = ngx_event_get_peer;
    p->upstream.log = s->connection->log;
    p->upstream.log_error = NGX_ERROR_ERR;

    /*  向上游的邮件服务器发起无阻塞的 TCP 连接  */
    rc = ngx_event_connect_peer(&p->upstream);

    if (rc == NGX_ERROR || rc == NGX_BUSY || rc == NGX_DECLINED) {
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    /*  需要监控接收邮件服务器的响应是否超时，于是把与上游间连接的读事件添加到
定时器中  */
    ngx_add_timer(p->upstream.connection->read, cscf->timeout);

    /*  设置连接的 data 成员指向 ngx_mail_session_t 结构体  */
    p->upstream.connection->data = s;
    p->upstream.connection->pool = s->connection->pool;

    /*  设置 nginx 与下游客户端间连接读事件的回调方法为不会读取内容的  */
    /* ngx_mail_proxy_block_read 方法，因为当前阶段 nginx 不会与客户端交互  */
    s->connection->read->handler = ngx_mail_proxy_block_read;

    /*  设置 nginx 与上游间的连接写事件回调方法为什么事都不做的  */
    /* ngx_mail_proxy_dummy_handler 方法，这意味着接下来向上游发送  */
    /* TCP 流时，将不再通过 epoll 这个事件框架来调度  */
    p->upstream.connection->write->handler = ngx_mail_proxy_dummy_handler;

    pcf = ngx_mail_get_module_srv_conf(s, ngx_mail_proxy_module);
```

```
    /* 建立 nginx 与邮件服务器间的内存缓冲区，缓冲区大小由 nginx.conf 文件中的
proxy_buffer 配置项决定 */
    s->proxy->buffer = ngx_create_temp_buf(s->connection->pool,
                                           pcf->buffer_size);
    if (s->proxy->buffer == NULL) {
        ngx_mail_proxy_internal_server_error(s);
        return;
    }

    /* 设置 out 为空，表示将不会再通过 out 向客户端发送响应 */
    s->out.len = 0;

    /* BEGIN: Added by zhangcan, 2018/6/8 */
#if (NGX_MAIL_SSL)

    /* 设置 nginx 与上游邮件服务器 SSL 加密方式 */
    if (cscf->mail_ssl && (ngx_mail_proxy_set_ssl(s, cscf, p) != NGX_OK)) {
        ngx_mail_session_internal_server_error(s);
        return;
    }

    ngx_connection_t *c = p->upstream.connection;

    if (cscf->mail_ssl && (c->ssl == NULL)) {

        /* nginx 与上游邮件服务器连接 SSL 初始化 */
        if (ngx_mail_proxy_ssl_init_connection(s, c) != NGX_OK) {
            ngx_mail_proxy_internal_server_error(s);
        }
        return;
    }

#endif
    /* END:    Added by zhangcan, 2018/6/8 */

    /* 根据用户请求的协议设置实际的邮件认证处理函数 */
    switch (s->protocol) {

    case NGX_MAIL_POP3_PROTOCOL:
        p->upstream.connection->read->handler = ngx_mail_proxy_pop3_handler;
        s->mail_state = ngx_pop3_start;
        break;

    case NGX_MAIL_IMAP_PROTOCOL:
```

```
            p->upstream.connection->read->handler = ngx_mail_proxy_imap_handler;
            s->mail_state = ngx_imap_start;
            break;

        default: /* NGX_MAIL_SMTP_PROTOCOL */
            p->upstream.connection->read->handler = ngx_mail_proxy_smtp_handler;
            s->mail_state = ngx_smtp_start;
            break;
        }
}
```

- 以下三个函数，ngx_mail_proxy_set_ssl、ngx_mail_proxy_ssl_init_connection、ngx_mail_proxy_ssl_handshake_handle 主要用于 nginx 与上游服务器以 SSL 方法建立连接及处理功能。

```
/**************************************************************************
*
    函 数 名 : ngx_mail_proxy_set_ssl
    功能描述 : 设置 nginx 与上游邮件服务器 SSL 加密方式
    输入参数 : ngx_mail_session_t *s
               ngx_mail_core_srv_conf_t *cscf
               ngx_mail_proxy_ctx_t *p
    输出参数 : 无
    返 回 值 : ngx_int_t
    作    者 : zhangcan
    日    期 : 2018 年 6 月 8 日
**************************************************************************
*/
static ngx_int_t
ngx_mail_proxy_set_ssl(ngx_mail_session_t *s,
    ngx_mail_core_srv_conf_t *cscf, ngx_mail_proxy_ctx_t *p)
{
    ngx_pool_cleanup_t    *cln;

    p->ssl = ngx_pcalloc(s->connection->pool, sizeof(ngx_ssl_t));
    if (p->ssl == NULL) {
        return NGX_ERROR;
    }

    p->ssl->log = s->connection->log;

    if (ngx_ssl_create(p->ssl, cscf->mail_ssl_protocols, NULL)
        != NGX_OK) {
        goto failed;
    }
```

```
    if (SSL_CTX_set_cipher_list(p->ssl->ctx,
        (const char *) cscf->mail_ssl_ciphers.data) == 0) {
        ngx_ssl_error(NGX_LOG_ERR, s->connection->log, 0,
                      "SSL_CTX_set_cipher_list(\"%V\") failed",
                      &cscf->mail_ssl_ciphers);
        goto failed;
    }

    cln = ngx_pool_cleanup_add(s->connection->pool, 0);
    if (cln == NULL) {
        goto failed;
    }

    cln->handler = ngx_ssl_cleanup_ctx;
    cln->data = p->ssl;

    return NGX_OK;

failed:
    if (p->ssl->ctx) {
        ngx_ssl_cleanup_ctx(p->ssl->ctx);
    }

    ngx_pfree(s->connection->pool, p->ssl);

    return NGX_ERROR;
}
```

```
/*************************************************************************
*
    函 数 名 : ngx_mail_proxy_ssl_init_connection
    功能描述 : nginx 与上游邮件服务器连接 SSL 初始化
    输入参数 : ngx_mail_session_t *s
               ngx_connection_t *c
    输出参数 : 无
    返 回 值 : ngx_int_t
    作    者 : zhangcan
    日    期 : 2018 年 6 月 8 日
*************************************************************************
*/
static ngx_int_t
ngx_mail_proxy_ssl_init_connection(ngx_mail_session_t *s, ngx_connection_t *c)
{
```

```
    ngx_int_t                       rc;

    /* 创建 SSL 连接 */
    if (ngx_ssl_create_connection(s->proxy->ssl, c,
        NGX_SSL_BUFFER|NGX_SSL_CLIENT) != NGX_OK) {
        if (c->ssl) {
            if (ngx_ssl_shutdown(c) == NGX_AGAIN) {
                c->ssl->handler = ngx_mail_close_connection;
                return NGX_ERROR;
            }
        }
        return NGX_ERROR;
    }

    /* SSL 握手 */
    rc = ngx_ssl_handshake(c);
    if (rc == NGX_AGAIN) {
        c->ssl->handler = ngx_mail_proxy_ssl_handshake_handle;
    }

    ngx_mail_proxy_ssl_handshake_handle(c);

    return NGX_OK;
}
```

```
/**************************************************************************
*
    函 数 名 : ngx_mail_proxy_ssl_handshake_handle
    功能描述 : nginx 与上游邮件服务器 SSL 握手
    输入参数 : ngx_connection_t *c
    输出参数 : 无
    返 回 值 : 无
    作      者 : zhangcan
    日      期 : 2018 年 6 月 8 日
**************************************************************************
*/
static void
ngx_mail_proxy_ssl_handshake_handle(ngx_connection_t *c)
{
    ngx_mail_session_t              *s;

    s = c->data;

    if (c->ssl->handshaked) {
```

```
        c->write->handler = ngx_mail_proxy_dummy_handler;

        /* 根据用户请求的协议设置实际的邮件认证处理函数 */
        switch (s->protocol) {

        case NGX_MAIL_POP3_PROTOCOL:
            c->read->handler = ngx_mail_proxy_pop3_handler;
            s->mail_state = ngx_pop3_start;
            break;

        case NGX_MAIL_IMAP_PROTOCOL:
            c->read->handler = ngx_mail_proxy_imap_handler;
            s->mail_state = ngx_imap_start;
            break;

        default: /* NGX_MAIL_SMTP_PROTOCOL */
            c->read->handler = ngx_mail_proxy_smtp_handler;
            s->mail_state = ngx_smtp_start;
            break;
        }
    }
}
```

9 源码编译

9.1 Nginx 源码编译

./configure --prefix=/usr/local/tengine --with-ipv6 --with-mail --with-mail_ssl_module --with-debug	
参数	作用
--prefix	设置安装路径，默认安装到/usr/local，当前安装到/usr/local/tengine。
--with-ipv6	使能 ipv6 功能
--with-mail	使能 POP3/IMAP/SMTP 代理功能。
--with-mail_ssl_module	使能 SSL 功能，这个主要指下游与 nginx 之间 SSL 功能。
--with-debug	使能 debug 日志，如果开启程序运行过程会在日志文件记录，调试日志，如果发布版时，可以关闭此功能。

9.2 PHP 源码编译

./configure --prefix=/usr/local/php --with-zlib --enable-zip --with-curl --with-openssl --with-sqlite3 --enable-fpm --enable-mbstring --enable-mysqlnd --with-mysqli --with-pdo-mysql --disable-phar	
参数	作用
--prefix	设置安装路径，默认安装到/usr/local，当前安装到/usr/local/php。
--with-zlib	使能 zlib 功能
--with-zip	使能 zip 功能。
--with-curl	使能 curl 功能。
--with-openssl	使能 SSL 功能。
--with-sqlite3	使能 sqlite3 功能。
--enable-fpm	使能 php-fpm 功能，主要用于解析 PHP 文件。
--enable-mbstring	使能多字节字符串功能。
--enable-mysqlnd	使能 php 的 mysql 驱动功能。
--with-mysqli --with-pdo-mysql	使能 mysql 功能。
说明：在编译完成后需要设置配置文件，具体步骤如下所示： 1、 cp /opt/zc/open_src/php/php-7.0.30/php.ini-production /usr/local/php/etc/php.ini 2、 cp /usr/local/php/etc/php-fpm.conf.default /usr/local/php/etc/php-fpm.conf 3、 cp /usr/local/php/etc/php-fpm.d/www.conf.default /usr/local/php/etc/php-fpm.d/www.conf 4、 cp /opt/zc/open_src/php/php-7.0.30/sapi/fpm/init.d.php-fpm /usr/local/php/sbin/init.d.php-fpm 通过执行相关命令 Usage: ./init.d.php-fpm {start|stop|force-quit|restart|reload|status|configtest} 来启动 php-fpm 进程。	

10 证书制作

步骤	命令	作用
第一步	openssl genrsa -out server.key 2048	生成根证书的私钥（key 文件）
第二步	openssl req -new -key server.key -out server.csr	利用私钥生成一个根证书的申请，一般证书的申请格式都是 csr，所以私钥和 csr 一般需要保存好。
第三步	openssl x509 -req -days 3650 -in server.csr -signkey server.key -out server.crt	自签名的方式签发我们之前的申请的证书，生成的证书为 server.crt，执行命令后打印如下信息： Signature ok subject=/C=CN/ST=Zhejiang/L=Hangzhou/O=SpinoCompany Corp./emailAddress=marketing@shipingin

		fo.com Getting Private key

说明：在执行第二步时，会让填写一些信息如下图所示。

```
[root@localhost 2]# openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:CN                    <- 国家代号，中国输入CN
State or Province Name (full name) []:Zhejiang          <- 省的全名，拼音
Locality Name (eg, city) [Default City]:Hangzhou        <- 市的全名，拼音
Organization Name (eg, company) [Default Company Ltd]:SpinoCompany Corp.   <- 公司英文名
Organizational Unit Name (eg, section) []:              <- 可以不输入
Common Name (eg, your name or your server's hostname) []:   <- 这输入要申请网站域名
Email Address []:marketing@shipinginfo.com              <- 电子邮箱，可随意填

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:                                <- 可以不输入
An optional company name []:                            <- 可以不输入
[root@localhost 2]#
```

11 结论

经过这几天研究及实验，发现 Nginx 支持 POP3、SMTP、IMAP 代理。但是存在一些问题，请关注第 12 章。

12 问题及解决方法

12.1 Nginx 与 PHP 进行认证存在大量 TIME_WAIT 连接，导致系统端口资源不够使用

解决方法：修改系统配置文件/etc/sysctl.conf 配置文件，修改如下所示。

```
# 表示开启SYN Cookies。当出现SYN等待队列溢出时，启用Cookies来处理，可防范少量

SYN攻击，默认为0，表示关闭

net.ipv4.tcp_syncookies = 1

# 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭

net.ipv4.tcp_tw_reuse = 1
```

```
# 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭
net.ipv4.tcp_tw_recycle=1
# 表示如果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2状态的时间
net.ipv4.tcp_fin_timeout=30
# 表示当keepalive起用的时候，TCP发送keepalive消息的频度。缺省是2小时，改为30分钟
net.ipv4.tcp_keepalive_time = 1800
# 表示用于向外连接的端口范围。缺省情况下很小：32768到61000，改为1024到65000
net.ipv4.ip_local_port_range = 1024 65000
# 表示SYN队列的长度，默认为1024，加大队列长度为8192，可以容纳更多等待连接的网络
连接
net.ipv4.tcp_max_syn_backlog = 8192
# 表示系统同时保持TIME_WAIT套接字的最大数量，如果超过这个数字，TIME_WAIT套接
字将立刻被清除并打印警告信息。
# 默认为180000，改为5000。对于Apache、Nginx等服务器，上几行的参数可以很好地减少
TIME_WAIT套接字数量
net.ipv4.tcp_max_tw_buckets = 5000
```

12.2 对于上游邮件服务器是 SSL 模式时，Nginx 邮件代理无法进行代理

目前 Nginx 邮件代理模块不支持与上游服务器之间采用 SSL 连接方式。通过添加部分代码已经支持 Nginx 与上游邮件服务器之间 SSL 连接方式。

12.3 对于下游客户端发送的数据需要自己解析

例如，需要获取发件人、收件人、邮件内容等，需要自己对发送数据进行解析。因为 Nginx 只对下游客户端与上游邮件服务器数据做透传。