ngx_stream_access_module 模块解析

ngx_stream_access_module 模块在 stream 子系统中是用来实现对某个 ip 访问控制功能。在 stream 子系统中，虽然没有像 http 子系统那样在代码中明确地定义处理阶段，但是其处理流程也是按照一定的阶段来划分的，stream 处理阶段包括：Post-accept、Pre-access、Access、SSL、Preread、Content、Log。

按照上面的阶段划分，ngx_stream_access_module 模块就是处在 Access 阶段的，下面便结合代码来分析下该模块的实现。

1、 配置命令解析

ngx_stream_access_module 模块目前支持两条配置指令，allow 和 deny。两个命令的使用方法是一样的，如下：

Syntax: allow/deny *address* | *CIDR* | unix: | all;

Default: —

Context: stream, server

从上面的用法可以看出，allow/deny 分别是用来允许/限制某个地址的访问，地址的格式可以是普通的 ip 地址、CIDR 形式的地址以及 Unix-domain sockets。如果是 all 的话，则表示允许/限制所有地址的访问。如果在配置文件中配置了 allow 或者 deny 命令，当配置文件解析模块解析到这两个命令的时候，Nginx 内部是如何来封装这两个命令的配置参数以便来实现其功能的呢？先来看下 Nginx 对这两个命令的定义：

```
static ngx_command_t    ngx_stream_access_commands[] = {

    { ngx_string("allow"),
      NGX_STREAM_MAIN_CONF|NGX_STREAM_SRV_CONF|NGX_CONF_TAKE1,
      ngx_stream_access_rule,
      NGX_STREAM_SRV_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("deny"),
      NGX_STREAM_MAIN_CONF|NGX_STREAM_SRV_CONF|NGX_CONF_TAKE1,
      ngx_stream_access_rule,
      NGX_STREAM_SRV_CONF_OFFSET,
      0,
      NULL },

      ngx_null_command
};
```

从这里可以看出，当配置文件中出现 allow/deny 命令的时候，都是调用 ngx_stream_access_rule 函数来解析其配置信息的。所以现在来看下这个函数的实现，为了简化分析，这里只看 ipv4 的处理（ipv6 和 unix-domain socket 的实现是类似的）。

```
static char *
ngx_stream_access_rule(ngx_conf_t *cf, ngx_command_t *cmd, void *conf)
{
    ngx_stream_access_srv_conf_t *ascf = conf;

    ngx_int_t                     rc;
    ngx_uint_t                    all;
    ngx_str_t                    *value;
    ngx_cidr_t                    cidr;
    ngx_stream_access_rule_t     *rule;
    ……
    ngx_memzero(&cidr, sizeof(ngx_cidr_t));
    /*  获取配置文件的配置参数  */
    value = cf->args->elts;

    /*  判断 allow 参数是不是"all"，如果是的话则 all 为 1 */
    all = (value[1].len == 3 && ngx_strcmp(value[1].data, "all") == 0);
    if (!all) {
        ……
        rc = ngx_ptocidr(&value[1], &cidr); //  计算 ip 地址和掩码
    }

    /* ipv4 */
    if (cidr.family == AF_INET || all) {

        /*  创建存储可访问和禁止访问规则的动态数组  */
        if (ascf->rules == NULL) {
            ascf->rules = ngx_array_create(cf->pool, 4,
                                           sizeof(ngx_stream_access_rule_t));
            if (ascf->rules == NULL) {
                return NGX_CONF_ERROR;
            }
        }

        /*  从动态数组中申请一个元素  */
        rule = ngx_array_push(ascf->rules);
        if (rule == NULL) {
            return NGX_CONF_ERROR;
        }

        /*  将 allow 或 deny 中配置的 ip 信息记录下来  */
        rule->mask = cidr.u.in.mask;
        rule->addr = cidr.u.in.addr;
        rule->deny = (value[0].data[0] == 'd') ? 1 : 0;
```

```
    }
    ……
    return NGX_CONF_OK;
}
```

在这里的实现可以看出，Nginx 内部将 allow/deny 命令抽象成了一条规则，其定义如下：

```
typedef struct {
    in_addr_t               mask;   // 掩码地址
    in_addr_t               addr;   // ip 地址
    /* 因为这个结构体是 allow 和 deny 共用，所以需要一个标志位来区分 */
    ngx_uint_t              deny;         /* unsigned   deny:1; */
} ngx_stream_access_rule_t;
```

所以解析 allow/deny 命令其实就是填充规则对象，并将规则对象挂载到了 ngx_stream_access_module 模块在 server 级别的配置项结构体里面，对于同一个 server 下面的 allow/deny 命令，则是以一个动态数组进行组织的。

在具体解析 allow/deny 命令的配置参数时，首先判断配置参数是不是 all，如果是的话，就不进行 ip 地址的解析转换，而是直接填充上面的规则。由于函数一开始就将 ngx_cidr_t 类型的对象 cidr 清零，所以如果参数是 all 的话，对应的规则中的 addr 和 mask 都是 0，而用来区分是 allow 还是 deny 命令的标志则视具体情况而定。如果配置参数不是 all 的话，那么就需要对配置参数进行转换，即把配置参数中的 ip 地址或者 cidr 形式的地址转换成对应的网络序值并存储在 ngx_cidr_t 对象中，然后从 ngx_stream_access_module 模块在 server 级别的配置项结构体的 rule 动态数组中申请一个元素，并将配置参数转换结果填充到规则中去。所以，如果一个 server 块中配置了多条 allow/deny 指令的话，等到 server 块解析完毕之后，那么这些命令参数都解析并存储到了 ngx_stream_access_module 模块在 server 级别的配置项结构体的 rule 动态数组中。

2、 访问控制介入

如果配置文件中某个 server 块内配置了 allow/deny 指令，在 Nginx 完成配置文件解析并提供 stream 四层反向代理服务后，客户端向 Nginx 中的该 server 发送请求时 Nginx 就会进行访问控制了，那么 Nginx 是通过什么方式来介入到四层方向代理流程中的呢？在 stream 框架重要组成部分--ngx_stream_core_module 模块的 stream main 级别配置项结构体中可以发现 Nginx 的处理方式，该配置项结构体如下：

```
typedef struct {
    /*
     *servers 动态数组存放的是代表出现在 stream 块内的 server 块的
     *配置项结构体，在 ngx_stream_core_server 函数中会将生成的
     *ngx_stream_core_module 模块的 srv 级别配置项结构体添加到
     *这个动态数组中。
     */
    ngx_array_t                 servers;         /* ngx_stream_core_srv_conf_t */

    /*
```

```
    * 存放的是 stream 块内所有 server 块内出现的 listen 指令的参数，
    * 一个 listen 对应其中的一个元素
    */
    ngx_array_t                 listen;         /* ngx_stream_listen_t */

    /* stream limit conn 模块注册的处理函数 */
    ngx_stream_access_pt       limit_conn_handler;

    /* stream access 模块注册的处理函数 */
    ngx_stream_access_pt       access_handler;
} ngx_stream_core_main_conf_t;
```

从这个配置项结构体中可以发现 Nginx 会将 ngx_stream_access_module 模块的处理函数挂载到这里。那 Nginx 是什么时候把 ngx_stream_access_module 模块的处理函数注册到这里的呢，又是什么时候会调用这个函数呢？

先来看下 Nginx 什么时候会把 ngx_stream_access_module 模块的处理函数注册到 ngx_stream_core_main_conf_t 对象的 access_handler 中的。ngx_stream_access_module 模块实现的 NGX_STREAM_MODULE 模块类型的接口 ngx_stream_module_t 如下：

```
static ngx_stream_module_t    ngx_stream_access_module_ctx = {
    ngx_stream_access_init,                 /* postconfiguration */

    NULL,                                   /* create main configuration */
    NULL,                                   /* init main configuration */

    ngx_stream_access_create_srv_conf,      /* create server configuration */
    ngx_stream_access_merge_srv_conf        /* merge server configuration */
};
```

从上面这个结构体中我们可以看到，ngx_stream_access_module 模块注册了 postconfiguration 字段的回调函数，而这个回调函数正是在解析完配置文件之后被调用的，该模块正是在这个回调函数中将其对应的处理函数挂载到了 ngx_stream_core_main_conf_t 对象的 access_handler 中，这从函数 ngx_stream_access_init 就可以看到，函数实现如下：

```
static ngx_int_t
ngx_stream_access_init(ngx_conf_t *cf)
{
    ngx_stream_core_main_conf_t  *cmcf;

    cmcf = ngx_stream_conf_get_module_main_conf(cf, ngx_stream_core_module);
    /* 注册 access_handler */
    cmcf->access_handler = ngx_stream_access_handler;

    return NGX_OK;
}
```

再来看下 Nginx 又是什么时候会调用 ngx_stream_access_module 模块的处理

函数来实现访问控制功能的。因为要对某个 ip 地址实现访问控制，所以 Nginx 必须得先和这个地址建立连接，但又必须要在提供具体的服务之前，否则访问控制就没有意义。所以根据访问控制的定位，ngx_stream_core_main_conf_t 结构体中的 access_handler 会回调函数会在 ngx_stream_init_connection 函数中被调用。而函数 ngx_stream_init_connection 则是在 Nginx 与客户端建立连接之后被调用，此时还没有提供任何服务，这与访问控制的思路是一致的。access_handler 回调函数被调用的地方如下：

```
void
ngx_stream_init_connection(ngx_connection_t *c)
{
    ……
    /*
     * 如果配置了 access 模块命令，则进行准入判断，如果禁止访问，
     * 则结束请求
     */
    if (cmcf->access_handler) {
        rc = cmcf->access_handler(s);

        if (rc != NGX_OK && rc != NGX_DECLINED) {
            ngx_stream_close_connection(c);
            return;
        }
    }
    ……
}
```

从这里可以看到 access_handler 如果返回的不是 NGX_OK 或者 NGX_DECLINED 的话，那么就会结束客户端请求及连接，也就是禁止该请求的的访问。

3、 访问控制实现

通过访问控制介入小节的分析，我们已经知道 ngx_stream_access_module 模块如何介入到 Nginx 的 stream 子系统的处理流程中。当客户端发来请求之后，Nginx 又是如何依据从配置文件中解析得到的规则来进行访问控制的呢？仍以 ipv4 为例，结合 ngx_stream_access_module 模块处理函数 ngx_stream_access_handler 来看下：

```
static ngx_int_t
ngx_stream_access_handler(ngx_stream_session_t *s)
{
    struct sockaddr_in                  *sin;
    ngx_stream_access_srv_conf_t    *ascf;
    ……
    ascf = ngx_stream_get_module_srv_conf(s, ngx_stream_access_module);

    /* 判断 nginx 与客户端之间的连接的协议族 */
    switch (s->connection->sockaddr->sa_family) {
```

```
        case AF_INET:
            /* ascf->rules 不为空说明在配置文件中配置了 access 规则 */
            if (ascf->rules) {
                /* 获取客户端地址信息 */
                sin = (struct sockaddr_in *) s->connection->sockaddr;
                return ngx_stream_access_inet(s, ascf, sin->sin_addr.s_addr);
            }
            break;
            ……
        }
        return NGX_DECLINED;
}
```

ngx_stream_access_handler 函数中调用的 ngx_stream_access_inet 函数实现如下：

```
static ngx_int_t
ngx_stream_access_inet(ngx_stream_session_t *s,
        ngx_stream_access_srv_conf_t *ascf, in_addr_t addr)
{
        ngx_uint_t                          i;
        ngx_stream_access_rule_t    *rule;

        /* 遍历解析配置文件时得到的规则动态数组，看 addr 是否在规则之中 */
        rule = ascf->rules->elts;
        for (i = 0; i < ascf->rules->nelts; i++) {

            /* 如果在规则数组中找到了对应的 ip 地址，则需要进一步处理 */
            if ((addr & rule[i].mask) == rule[i].addr) {
                return ngx_stream_access_found(s, rule[i].deny);
            }
        }

        /*
         * 如果客户端 ip 地址不在规则数组中，则返回 NGX_DECLINED，
         * 主流程往下继续执行
         */
        return NGX_DECLINED;
}
```

Ngx_stream_access_inet 函数中调用的 ngx_stream_access_found()函数实现如下：

```
static ngx_int_t
ngx_stream_access_found(ngx_stream_session_t *s, ngx_uint_t deny)
{
        /* 如果是 deny 的话，则返回 NGX_ABORT，表明当前 ip 不允许访问 */
        if (deny) {
```

```
            ……
            return NGX_ABORT;
        }

        /* 如果是 allow，则返回 NGX_OK，表明当前 ip 允许访问 */
        return NGX_OK;
}
```

从上面三个函数的实现我们可以非常清楚地看到 ngx_stream_access_module 模块实现的访问控制功能，其判断逻辑如下：

1、如果配置文件中没有配置访问控制规则，即没有配置 allow/deny 命令，则不对客户端请求进行访问控制。

2、如果触发此次请求的客户端 ip 地址不在规则之内，则不对该 ip 地址进行访问控制判断，并返回 NGX_DECLINED 触发 ngx_stream_init_connection()中的主流程继续往后续阶段处理。

3、如果触发此次请求的客户端 ip 地址在规则之内，并且该规则对应的是 allow 指令的话，则允许该 ip 地址进行访问，返回 NGX_OK 触发主流程往后续阶段处理，如果规则对应的是 deny 指令的话，则不允许该规则访问，返回 NGX_ABORT，触发主流程结束请求的后续处理。

以上便是 Nginx 访问控制功能的实现机制。