

fastdfs-nginx扩展模块源码分析

FastDFS-Nginx扩展模块源码分析

1. 背景

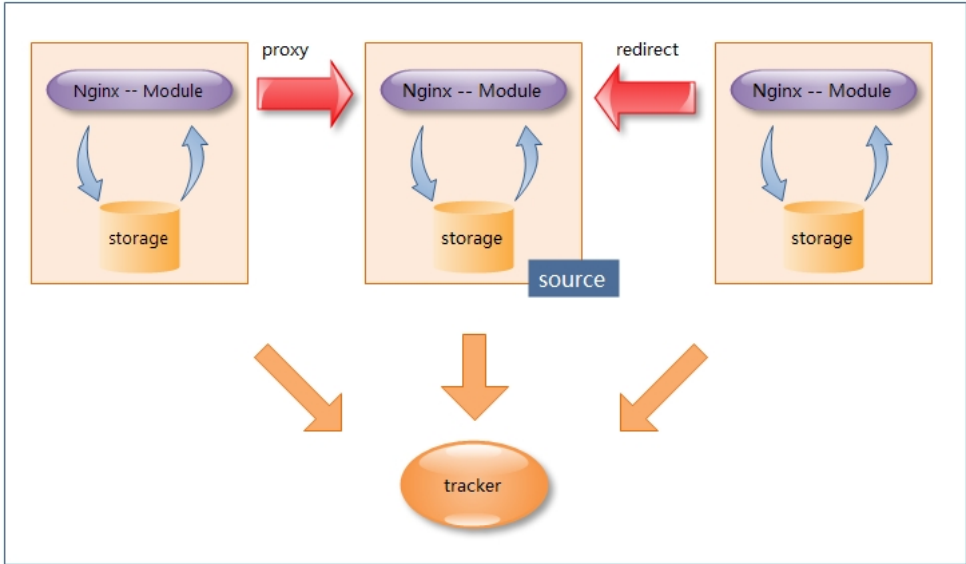
在大多数业务场景中，往往需要为FastDFS存储的文件提供http下载服务，而尽管FastDFS在其storage及tracker都内置了http服务，但性能表现却不尽如人意；作者余庆在后来的版本中增加了基于当前主流web服务器的扩展模块(包括nginx/apache)，其用意在于利用web服务器直接对本机storage数据文件提供http服务，以提高文件下载的性能。

2. 概要介绍

关于FastDFS的架构原理不再赘述，有兴趣可以参考：<http://code.google.com/p/fastdfs/wiki/Overview>

2.1 参考架构

使用FastDFS整合Nginx的参考架构如下所示



说明：在每一台storage服务器主机上部署Nginx及FastDFS扩展模块，由Nginx模块对storage存储的文件提供http下载服务，仅当前storage节点找不到文件时会向源storage主机发起redirect或proxy动作。
注：图中的tracker可能为多个tracker组成的集群；且当前FastDFS的Nginx扩展模块支持单机多个group的情况

2.2 几个概念

storage_id：指storage server的id，从FastDFS4.x版本开始，tracker可以对storage定义一组ip到id的映射，以id的形式对storage进行管理。而文件名写入的不再是storage的ip而是id，这样的方式对于数据迁移十分有利。
storage_sync_file_max_delay：指storage节点同步一个文件最大的时间延迟，是一个阈值；如果当前时间与文件创建时间的差距超过该值则认为同步已经完成。
anti_steal_token：指文件ID防盗链的方式，FastDFS采用token认证的方式进行文件防盗链检查。

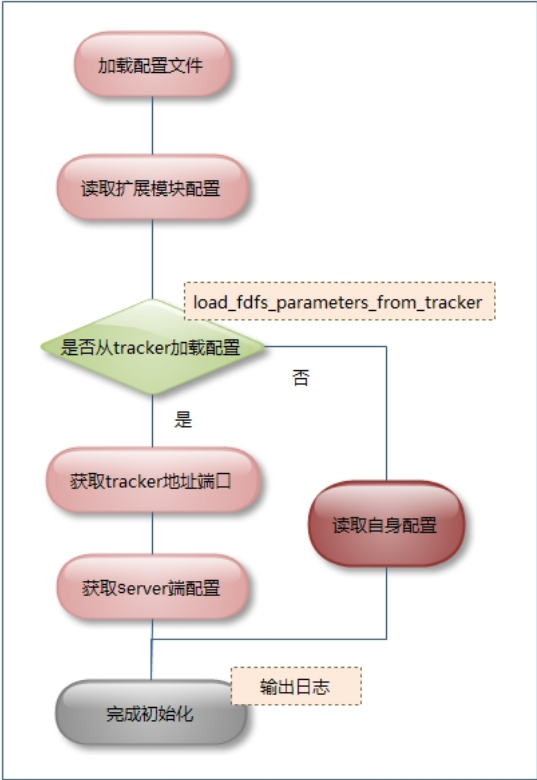
3. 实现原理

3.1 源码包说明

下载后的源码包很小，仅包括以下文件：

```
ngx_http_fastdfs_module.c //nginx-module接口实现文件，用于接入fastdfs-module核心模块逻辑
common.c //fastdfs-module核心模块，实现了初始化、文件下载的主要逻辑
common.h //对应于common.c的头文件
config //编译模块所用的配置，里面定义了一些重要的常量，如扩展配置文件路径、文件下载chunk大小
mod_fastdfs.conf //扩展配置文件的demo
```

3.2 初始化



3.2.1 加载配置文件

目标文件: /etc/fdfs/mod_fastdfs.conf

3.2.2 读取扩展模块配置

一些重要参数包括:

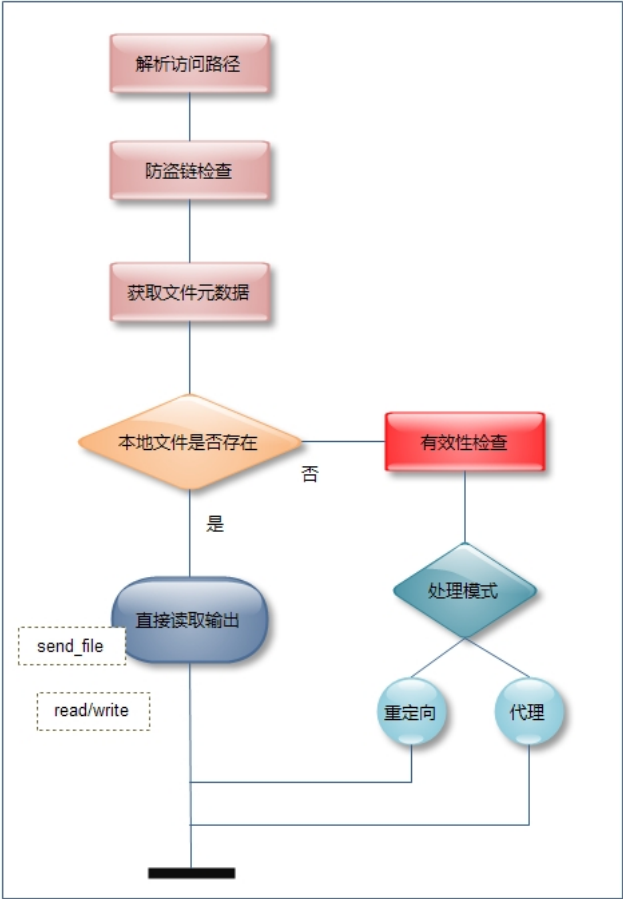
```
group_count           //group个数
url_have_group_name   //url中是否包含group
group.store_path       //group对应的存储路径
connect_timeout        //连接超时
network_timeout        //接收或发送超时
storage_server_port    //storage_server端口, 用于在找不到文件情况下连接源storage下载文件 (该做法已过时)
response_mode          //响应模式, proxy或redirect
load_fdfs_parameters_from_tracker //是否从tracker下载服务端配置
```

3.2.3 加载服务端配置

根据load_fdfs_parameters_from_tracker参数确定是否从tracker获取server端的配置信息

- load_fdfs_parameters_from_tracker=true:
 - 调用fdfs_load_tracker_group_ex解析tracker连接配置;
 - 调用fdfs_get_ini_context_from_tracker连接tracker获取配置信息;
 - 获取storage_sync_file_max_delay阈值
 - 获取use_storage_id
 - 如果use_storage_id为true, 则连接tracker获取storage_ids映射表(调用方法:
fdfs_get_storage_ids_from_tracker_group)
- load_fdfs_parameters_from_tracker=false:
 - 从mod_fastdfs.conf加载所需配置: storage_sync_file_max_delay、use_storage_id;
 - 如果use_storage_id为true, 则根据storage_ids_filename获取storage_ids映射表(调用方法:
fdfs_load_storage_ids_from_file)

3.3 下载过程



3.3.1 解析访问路径

得到group和file_id_without_group两个参数;

3.3.2 防盗链检查

- 根据g_http_params.anti_steal_token配置(见http.conf文件), 判断是否进行防盗链检查;
- 采用token的方式实现防盗链, 该方式要求下载地址带上token, 且token具有时效性(由ts参数指明);

检查方式:

md5(fileid_without_group + privKey + ts) = token; 同时ts没有超过ttl范围 (可参考JavaClient CommonProtocol)

调用方法: fdfs_http_check_token

关于FastDFS的防盗链可参考: <http://bbs.chinaunix.net/thread-1916999-1-1.html>

3.3.3 获取文件元数据

根据文件ID 获取元数据信息, 包括: 源storage ip,文件路径、名称, 大小

代码:

```
if ((result=fdfs_get_file_info_ex1(file_id, false, &file_info)) != 0)...
```

在fdfs_get_file_info_ex1 的实现中, 存在一个取巧的逻辑:

当获得文件的ip段之后, 仍然需要确定该段落是storage的id还是ip.

代码:

```
fdfs_shared.func.c
-> fdfs_get_server_id_type(ip_addr.s_addr) == FDFS_ID_TYPE_SERVER_ID
...
if (id > 0 && id <= FDFS_MAX_SERVER_ID) {
    return FDFS_ID_TYPE_SERVER_ID;
} else {
    return FDFS_ID_TYPE_IP_ADDRESS;
}
```

判断标准为ip段的整数值是否在 0 到 $\text{FDFS_MAX_SERVER_ID}$ (见tracker_types.h)之间;

其中 $\text{FDFS_MAX_SERVER_ID} = (1 \ll 24) - 1$, 该做法利用了ipv4地址的特点(由4*8个二进制位组成), 即ipv4地址数值务必大于该阈值

3.3.4 检查本地文件是否存在

调用`trunk_file_stat_ex1`获取本地文件信息, 该方法将实现:

1. 辨别当前文件是trunkfile还是singlefile
2. 获得文件句柄fd
3. 如果文件是trunk形式则同时也将相关信息(偏移量/长度)一并获得

代码:

```
if (bSameGroup)
{
    FDFSTrunkHeader trunkHeader;
    if ((result=trunk_file_stat_ex1(pStorePaths, store_path_index, \
        true_filename, filename_len, &file_stat, \
        &trunkInfo, &trunkHeader, &fd)) != 0)
    {
        bFileExists = false;
    }
    else
    {
        bFileExists = true;
    }
}
else
{
    bFileExists = false;
    memset(&trunkInfo, 0, sizeof(trunkInfo));
}
```

3.3.5 文件不存在的处理

- 进行有效性检查

检查项有二:

A. 源storage是本机或者当前时间与文件创建时间的差距已经超过阈值, 报错;

代码:

```
if (is_local_host_ip(file_info.source_ip_addr) || \
    (file_info.create_timestamp > 0 && (time(NULL) - \
        file_info.create_timestamp > ''storage_sync_file_max_delay'')))
```

B. 如果是redirect后的场景, 同样报错;

如果是由其他storage节点redirect过来的请求, 其url参数中会存在redirect一项

在通过有效性检查之后将进行代理或重定向处理

- 重定向模式

配置项`response_mode = redirect`, 此时服务端返回返回302响应码, url如下:

```
http:// {源storage地址} : {当前port} {当前url} {参数"redirect=1"} (标记已重定向过)
```

代码:

```
response.redirect_url_len = snprintf( \
    response.redirect_url, \
    sizeof(response.redirect_url), \
    "http://%s%s%s%c%s", \
    file_info.source_ip_addr, port_part, \
    path_split_str, url, \
    param_split_char, "redirect=1");
```



注：该模式下要求源storage配备公开访问的webserver、同样的端口(一般是80)、同样的path配置。

- 代理模式

配置项response_mode = proxy，该模式的工作原理如同反向代理的做法，而**仅仅使用源storage地址作为代理proxy的host**，其余部分保持不变。

代码:

```
if (pContext->proxy_handler != NULL)
{
    return pContext->proxy_handler(pContext->arg, \
        file_info.source_ip_addr);
}

//其中proxy_handler方法来自ngx_http_fastdfs_module.c文件的ngx_http_fastdfs_proxy_handler方法
//其实现中设置了大量回调、变量，并最终调用代理请求方法，返回结果：
rc = ngx_http_read_client_request_body(r, ngx_http_upstream_init); //执行代理请求，并返回结果
```

3.3.6 输出本地文件

当本地文件存在时，将直接输出。

- 根据是否**trunkfile**获取文件名，文件名长度、文件offset;

代码:



```
bTrunkFile = IS_TRUNK_FILE_BY_ID(trunkInfo);
if (bTrunkFile)
{
    trunk_get_full_filename_ex(pStorePaths, &trunkInfo, \
        full_filename, sizeof(full_filename));
    full_filename_len = strlen(full_filename);
    file_offset = TRUNK_FILE_START_OFFSET(trunkInfo) + \
        pContext->range.start;
}
else
{
    full_filename_len = sprintf(full_filename, \
        sizeof(full_filename), "%s/data/%s", \
        pStorePaths->paths[store_path_index], \
        true_filename);
    file_offset = pContext->range.start;
}
```



- 若nginx开启了send_file开关而且当前为非chunkFile的情况下尝试使用**sendfile**方法以优化性能;

代码:




```
if (pContext->send_file != NULL && !bTrunkFile)
{
    http_status = pContext->if_range ? \
        HTTP_PARTIAL_CONTENT : HTTP_OK;
    OUTPUT_HEADERS(pContext, (&response), http_status)
    .....
    return pContext->send_file(pContext->arg, full_filename, \
        full_filename_len, file_offset, download_bytes);
}
```



- 否则使用lseek 方式随机访问文件，并输出相应的段;


做法：使用chunk方式循环读，输出...

代码:



```
while (remain_bytes > 0)
{
    read_bytes = remain_bytes <= FDFS_OUTPUT_CHUNK_SIZE ? \
        remain_bytes : FDFS_OUTPUT_CHUNK_SIZE;
    if (read(fd, file_trunk_buff, read_bytes) != read_bytes)
    {
        close(fd);
        .....
        return HTTP_INTERNAL_SERVER_ERROR;
    }

    remain_bytes -= read_bytes;
    if (pContext->send_reply_chunk(pContext->arg, \
        (remain_bytes == 0) ? 1: 0, file_trunk_buff, \
        read_bytes) != 0)
    {
        close(fd);
        return HTTP_INTERNAL_SERVER_ERROR;
    }
}
```



其中chunk大小见config文件配置: **-DFDFS_OUTPUT_CHUNK_SIZE='256*1024'**

4. 扩展阅读

基于Referer实现防盗链:

<http://www.cnblogs.com/wjiang/archive/2010/04/04/1704445.html>

FastDFS使用FAQ:

<http://bbs.chinaunix.net/thread-1920470-1-1.html>

FastDFS-Nginx扩展的配置参考:

<http://blog.csdn.net/poechant/article/details/7036594>

FastDFS配置、部署资料整理-CSDN博客:

<http://blog.csdn.net/poechant/article/details/6996047>

关于C语言open和fopen区别

<http://blog.csdn.net/hairetz/article/details/4150193>