FastDFS

https://www.cnblogs.com/1477717815fuming/p/8404882.html

https://zhuanlan.zhihu.com/p/61735659

一、课前准备

- 下载 FastDFS 安装包
- 下载 Nginx 安装包
- 下载 FastDFS 与 Nginx 的整合模块

二、FastDFS 安装

```
#全部依赖环境安装
yum -y install zlib zlib-devel pcre pcre-devel gcc gcc-c++ openssl openssl-devel libevent
libevent-devel perl unzip net-tools wget
```

1、 下载

- 下载 libfastcommon 包（github 源码）

  `https://github.com/happyfish100/libfastcommon/releases`

  - linux wget下载

    ```
    wget https://github.com/happyfish100/libfastcommon/archive/V1.0.39.tar.gz
    ```

- 下载 fastdfs 源码包（github 源码）

  `https://github.com/happyfish100/fastdfs/releases`

  - linux wget下载

    ```
    wget https://github.com/happyfish100/fastdfs/archive/V5.11.tar.gz
    ```

2、 需求

- Tracker Server：192.168.10.135
- Storage Server：192.168.10.136、192.168.10.137

3 、安装

3.1.tracker 和 storage 安装

`tracker server和storage server都有一些相同的安装操作，如下：`

- **安装 gcc 环境**

```
yum install -y gcc-c++
```

- **安装 libevent**，FastDFS 依赖 libevent 库（暂不安装）

```
yum install -y libevent
```

- **安装 libfastcommon**，libfastcommon 是 FastDFS 官方提供的包，包含了 FastDFS 运行所需要的一些基础库。

```
wget https://github.com/happyfish100/libfastcommon/archive/V1.0.39.tar.gz

tar -zxvf  V1.0.39.tar.gz

cd libfastcommon-1.0.39
#如果系统中没有make指令，需要安装
yum -y install gcc automake autoconf libtool make

./make.sh && ./make.sh  install
```

- **拷贝 libfastcommon.so 文件至 /usr/lib 目录（新版本不需要此步）**

```
cp  /usr/lib64/libfastcommon.so  /usr/lib/
```

注：

libfastcommon安装好后会自动将库文件拷贝至/usr/lib64下，由于FastDFS程序引用usr/lib目录，所以需要将/usr/lib64下的库文件拷贝至/usr/lib下。

- 下载安装 FastDFS，进入 FastDFS 目录，编译安装

```
wget https://github.com/happyfish100/fastdfs/archive/V5.11.tar.gz

tar -zxvf  V5.11.tar.gz

cd cd fastdfs-5.11

./make.sh && ./make.sh install
```

- **拷贝/root/fastdfs-5.11/conf目录下的文件到/etc/fdfs目录下**

```
cp /kkb/soft/fastdfs-5.11/conf/* /etc/fdfs
```

3.2.tracker server配置

注意：base_path目录要存在

- 修改/etc/fdfs/tracker.conf

```
vim /etc/fdfs/tracker.conf
```

修改内容如下：

```
base_path=/kkb/server/fastdfs/tracker
```

- **创建tracker服务器上面的目录**

```
mkdir /kkb/server/fastdfs/tracker -p
```

3.3.storage server配置

注意：base_path和store_path0目录要存在

- 修改/etc/fdfs/storage.conf

```
vim /etc/fdfs/storage.conf
```

修改内容如下：

```
#指定storage的组名
group_name=group1
base_path=/kkb/server/fastdfs/storage
store_path0=/kkb/server/fastdfs/storage M00虚拟磁盘路径
#如果有多个挂载磁盘则定义多个store_path，如下
#store_path1=.....
#store_path2=......
#配置tracker服务器IP和端口
tracker_server=111.231.106.221:22122
#如果有多个则配置多个tracker
#tracker_Server=192.168.101.4:22122
```

- **创建storage服务器上面的目录**

```
mkdir /kkb/server/fastdfs/storage -p
```

4、 启动（启动前记得关闭防火墙）

5、 防火墙关闭

```
systemctl stop firewalld
```

6、 禁止防火墙开机启动

```
systemctl disable firewalld
```

6.1. Tracker 启动命令

```
/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
```

6.2. Storage 启动命令

```
/usr/bin/fdfs_storaged /etc/fdfs/storage.conf
```

7、 集群状态检查命令

```
fdfs_monitor /etc/fdfs/storage.conf
```

8、 删除 Storage

```
fdfs_monitor /etc/fdfs/storage.conf delete 组名 storage的ip
```

9、 Tracker 关闭命令

```
killall fdfs_trackerd
```

10、 Storage 关闭命令

```
killall fdfs_storaged
```

11、 缺少 killall 命令进行安装

```
yum install psmisc -y
```

11.1. Tracker 开机自启动

```
vim /etc/rc.d/rc.local
```

将运行命令行添加进文件：

```
/usr/bin/fdfs_trackerd /etc/fdfs/tracker.conf
```

11.2. Storage 开机自启动

```
vim /etc/rc.d/rc.local
```

将运行命令行添加进文件：

```
/usr/bin/fdfs_storaged /etc/fdfs/storage.conf
```

12、 上传图片测试

FastDFS安装成功后可通过【fdfs_test】命令测试上传、下载等操作。

- 修改client.conf

```
vim /etc/fdfs/client.conf
```

修改内容如下：

```
base_path=/kkb/server/fastdfs/client
tracker_Server=111.231.106.221:22122
```

- 创建 client 的数据目录

```
mkdir -p /kkb/server/fastdfs/client
```

- 使用 ==fdfs_test== 命令将/home下的tomcat.png上传到FastDFS中

```
/usr/bin/fdfs_test  /etc/fdfs/client.conf  upload  /etc/fdfs/anti-steal.jpg
```

说明：

http://192.168.10.135/group1/M00/00/00/wKhlBVVY2M-AM_9DAAAT7-0xdqM485_big.png就是文件的访问路径。

对应storage服务器上的磁盘路径：

/home/fastdfs/fdfs_storage/data/00/00/wKhlBVVY2M-AM_9DAAAT7-0xdqM485_big.png文件。

13、 tracker.conf

13.1. 基本配置

```
disable
#func: 配置是否生效
#valu: true、false
```

```
disable=false
bind_addr
#func: 绑定IP
#valu: IP地址
bind_addr=192.168.6.102
port
#func: 服务端口
#valu: 端口整数值
port=22122
connect_timeout
#func: 连接超时
#valu: 秒单位正整数值
connect_timeout=30
network_timeout
#func: 网络超时
#valu: 秒单位正整数值
network_timeout=60
base_path
#func: Tracker数据/日志目录地址
#valu: 路径
base_path=/home/michael/fdfs/base4tracker
max_connections
#func: 最大连接数
#valu: 正整数值
max_connections=256
work_threads
#func: 线程数，通常设置CPU数
#valu: 正整数值
work_threads=4
store_lookup
#func: 上传文件的选组方式。
#valu: 0、1或2。
# 0: 表示轮询
# 1: 表示指定组
# 2: 表示存储负载均衡（选择剩余空间最大的组）
store_lookup=2
store_group
#func: 指定上传的组，如果在应用层指定了具体的组，那么这个参数将不会起效。另外如果store_lookup如果是0或2，则此参数无效。
#valu: group1等
store_group=group1
store_Server
#func: 上传服务器的选择方式。（一个文件被上传后，这个storage Server就相当于这个文件的storage Server源，会对同组的storage Server推送这个文件达到同步效果）
#valu: 0、1或2
# 0: 轮询方式（默认）
# 1: 根据ip 地址进行排序选择第一个服务器（IP地址最小者）
# 2: 根据优先级进行排序（上传优先级由storage Server来设置，参数名为upload_priority），优先级值越小优先级越高。
store_Server=0
store_path
#func: 上传路径的选择方式。storage Server可以有多个存放文件的base path（可以理解为多个磁盘）。
#valu:
# 0: 轮流方式，多个目录依次存放文件
# 2: 存储负载均衡。选择剩余空间最大的目录存放文件（注意：剩余磁盘空间是动态的，因此存储到的目录或磁盘可能也是变化的）
store_path=0
download_Server
#func: 下载服务器的选择方式。
#valu:
```

```
# 0：轮询（默认）
# 1：IP最小者
# 2：优先级排序（值最小的，优先级最高。）
download_Server=0
reserved_storage_space
#func：保留空间值。如果某个组中的某个服务器的剩余自由空间小于设定值，则文件不会被上传到这个组。
#valu:
# G or g for gigabyte
# M or m for megabyte
# K or k for kilobyte
reserved_storage_space=1GB
log_level
#func：日志级别
#valu:
# emerg for emergency
# alert
# crit for critical
# error
# warn for warning
# notice
# info for information
# debug for debugging
log_level=info
run_by_group / run_by_user
#func：指定运行该程序的用户组
#valu：用户组名或空
run_by_group=

#func:
#valu:
run_by_user=
allow_hosts
#func：可以连接到tracker Server的ip范围。可设定多个值。
#valu
allow_hosts=
check_active_interval
#func：检测 storage Server 存活的时间隔，单位为秒。
#       storage Server定期向tracker Server 发心跳，
#       如果tracker Server在一个check_active_interval内还没有收到storage Server的一次心跳，
#       那边将认为该storage Server已经下线。所以本参数值必须大于storage Server配置的心跳时间间隔。
#       通常配置为storage Server心跳时间间隔的2倍或3倍。
check_active_interval=120
thread_stack_size
#func：设定线程栈的大小。 线程栈越大，一个线程占用的系统资源就越多。
#       如果要启动更多的线程（V1.x对应的参数为max_connections，V2.0为work_threads），可以适当降低本参数
值。
#valu：如64KB，默认值为64，tracker Server线程栈不应小于64KB
thread_stack_size=64KB
storage_ip_changed_auto_adjust
#func：这个参数控制当storage Server IP地址改变时，集群是否自动调整。注：只有在storage Server进程重启时才
完成自动调整。
#valu：true或false
storage_ip_changed_auto_adjust=true
```

13.2. 同步

```
storage_sync_file_max_delay
#func: 同组storage服务器之间同步的最大延迟时间。存储服务器之间同步文件的最大延迟时间，根据实际情况进行调整
#valu: 秒为单位，默认值为1天（24*3600）
#sinc: v2.0
storage_sync_file_max_delay=86400
storage_sync_file_max_time
#func: 存储服务器同步一个文件需要消耗的最大时间，缺省为300s，即5分钟。
#sinc: v2.0
storage_sync_file_max_time=300
sync_log_buff_interval
#func: 同步或刷新日志信息到硬盘的时间间隔。注意：tracker Server 的日志不是时时写硬盘的，而是先写内存。
#valu: 以秒为单位
sync_log_buff_interval=10
```

13.3. trunk 和 slot

```
#func: 是否使用trunk文件来存储几个小文件
#valu: true或false
#sinc: v3.0
use_trunk_file=false

#func: 最小slot大小
#valu: <= 4KB，默认为256字节
#sinc: v3.0
slot_min_size=256

#func: 最大slot大小
#valu: >= slot_min_size，当小于这个值的时候就存储到trunk file中。默认为16MB。
#sinc: v3.0
slot_max_size=16MB

#func: trunk file的size
#valu: >= 4MB，默认为64MB
#sinc: v3.0
trunk_file_size=64MB
```

13.4. HTTP相关

```
是否启用 HTTP
#func: HTTP是否生效
#valu: true或false
http.disabled=false
HTTP 服务器端口号
#func: tracker Server上的http port
#valu:
#note: 只有http.disabled=false时才生效
http.Server_port=7271
检查Storage存活状态的间隔时间（心跳检测）
#func: 检查storage http Server存活的间隔时间
#valu: 单位为秒
#note: 只有http.disabled=false时才生效
http.check_alive_interval=30
心跳检测使用的协议方式
#func: 检查storage http Server存活的方式
#valu:
# tcp: 连接到storage Server的http端口，不进行request和response。
# http: storage check alive url must return http status 200.
#note: 只有http.disabled=false时才生效
http.check_alive_type=tcp
```

```
检查 Storage 状态的 URI
#func: 检查storage http Server是否alive的uri/url
#note: 只有http.disabled=false时才生效
http.check_alive_uri=/status.html
```

三、FastDFS 的 Nginx 模块原理分析

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；(**余老师在 V4.05 以后的版本就把内置 HTTP服务去掉了**)。

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。

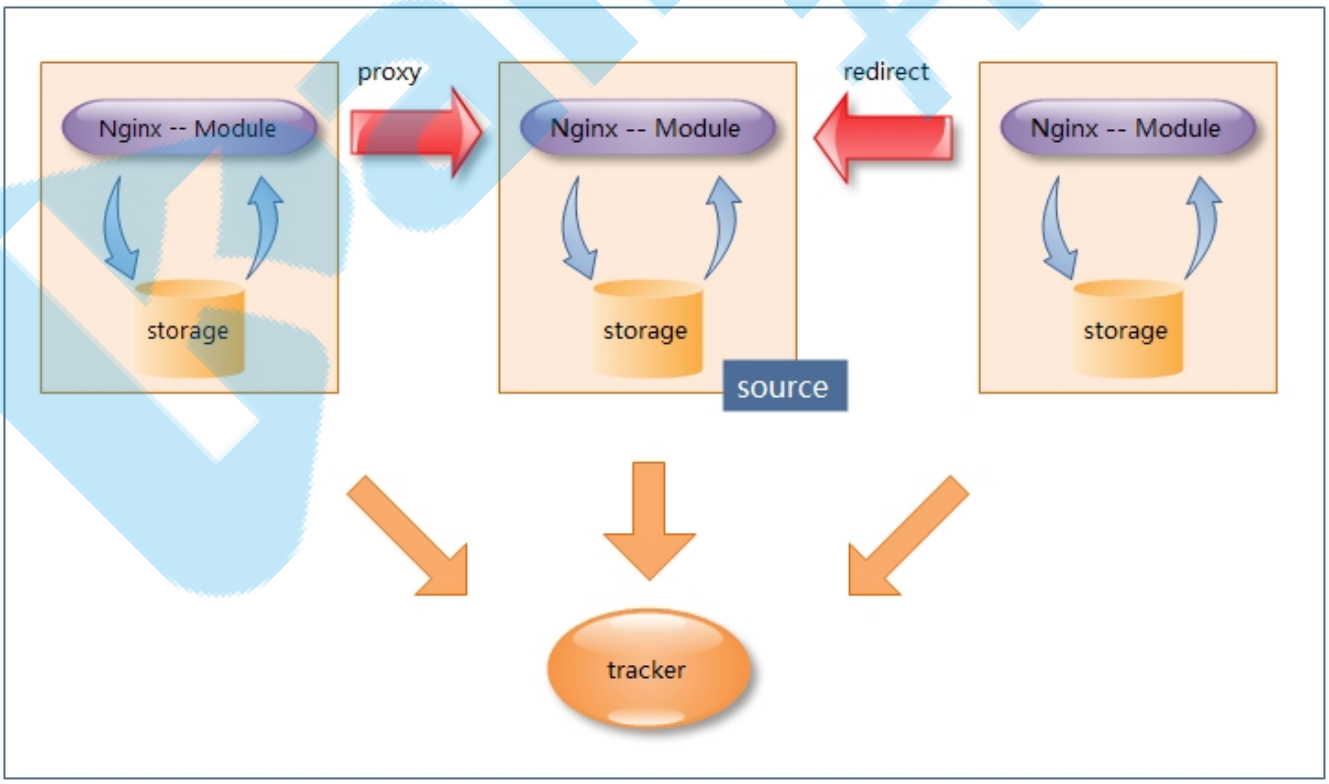
其实不使用Nginx的扩展模块，只安装web服务器（Nginx和Apache），也可以对存储的文件进行访问。那么为什么要使用Nginx的扩展模块来访问存储的文件，原因有两个：

- 如果进行文件合并存储，那么不使用FastDFS的nginx扩展模块，是无法访问到合并后的文件的，因为文件合并之后，多个小文件都是存储在一个trunk文件中的，在存储目录下，是看不到具体的小文件的。

- 如果文件未同步成功，那么不使用FastDFS的nginx扩展模块，是无法正常访问到指定的文件的，而使用了FastDFS的nginx扩展模块之后，如果要访问的文件未同步成功，那么会解析出来该文件的源存储服务器ip，然后将该访问请求重定向或者代理到源存储服务器中进行访问。

2、 概要介绍

2.1. FastDFS 整合 Nginx 的参考架构



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。

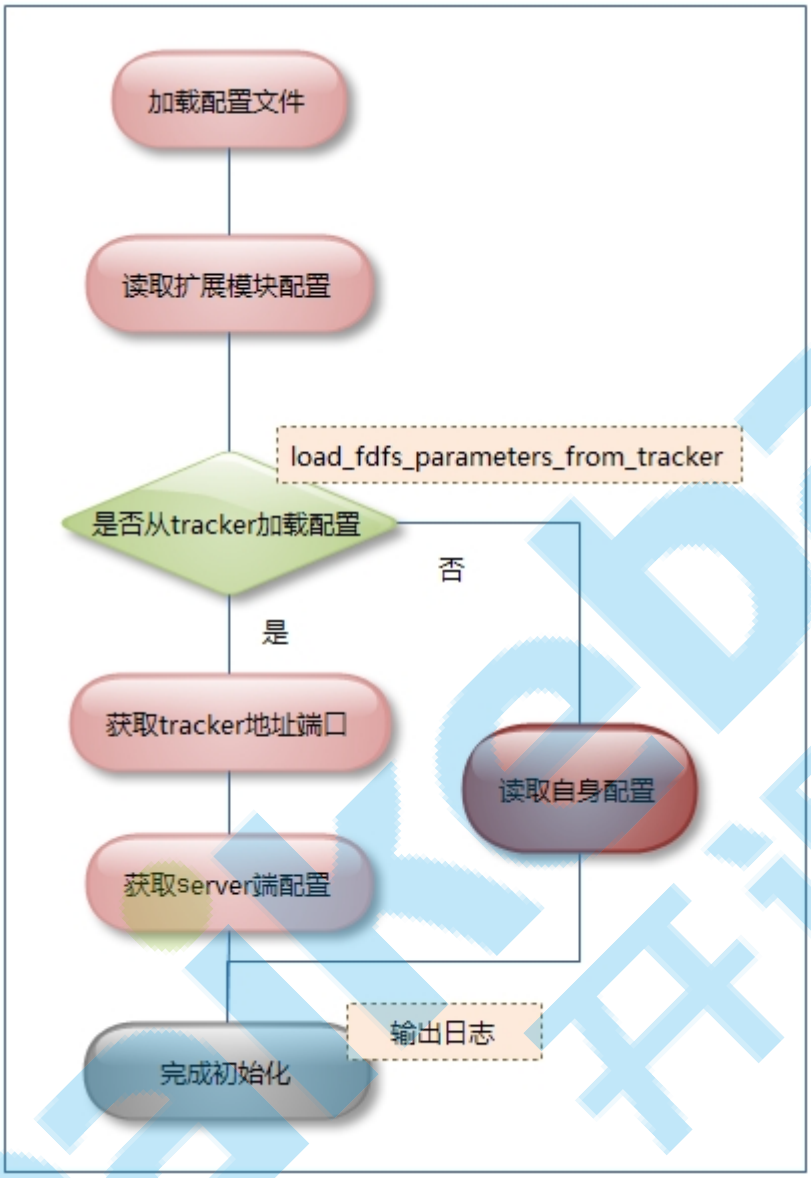
3、 实现原理

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



1）加载配置文件

目标文件：**/etc/fdfs/mod_fastdfs.conf**

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）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

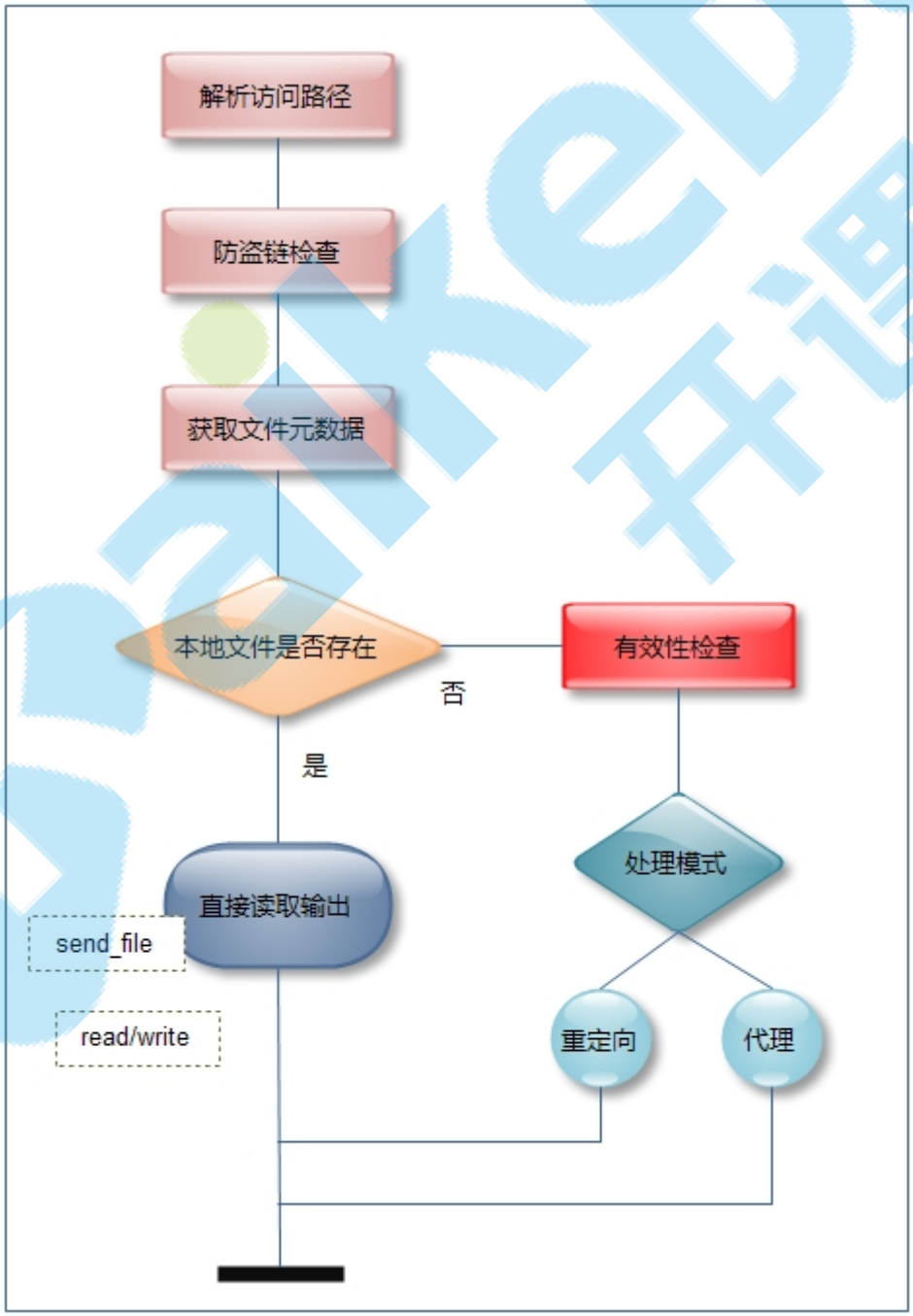
- **load_fdfs_parameters_from_tracker=true:**

1. 调用fdfs_load_tracker_group_ex解析tracker连接配置；
2. 调用fdfs_get_ini_context_from_tracker连接tracker获取配置信息；
3. 获取storage_sync_file_max_delay阈值
4. 获取use_storage_id
5. 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:**

1. 从mod_fastdfs.conf加载所需配置：storage_sync_file_max_delay、use_storage_id;
2. 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(重点)



解析访问路径

得到group和file_id_without_group两个参数;

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

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 -> **FDFS_MAX_SERVER_ID**(见tracker_types.h)之间； *其中FDFS_MAX_SERVER_ID = (1 << 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*

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));
    }
```

文件不存在的处理

- **进行有效性检查**

检查项有二：

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''')))
```

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%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
```

输出本地文件

- 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 = snprintf(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'**

四、配置FastDFS的Nginx模块

FastDFS的Nginx模块需要安装到每一台storage server中

下载文件

```
wget https://github.com/happyfish100/fastdfs-nginx-module/archive/V1.20.tar.gz
```

解压缩

```
tar -zxvf V1.20.tar.gz
```

修改config文件（特别关键的一步）

修改fastdfs-nginx-module/src/config文件

```
cd fastdfs-nginx-module-1.20/src
vim config
```

- **修改前的内容如下：**

```
ngx_addon_name=ngx_http_fastdfs_module

if test -n "${ngx_module_link}"; then
    ngx_module_type=HTTP
    ngx_module_name=$ngx_addon_name
    ngx_module_incs="/usr/local/include"
    ngx_module_libs="-lfastcommon -lfdfsclient"
    ngx_module_srcs="$ngx_addon_dir/ngx_http_fastdfs_module.c"
    ngx_module_deps=
    CFLAGS="$CFLAGS -D_FILE_OFFSET_BITS=64 -DFDFS_OUTPUT_CHUNK_SIZE='256*1024' -
DFDFS_MOD_CONF_FILENAME='\"/etc/fdfs/mod_fastdfs.conf\"'"
    . auto/module
else
    HTTP_MODULES="$HTTP_MODULES ngx_http_fastdfs_module"
    NGX_ADDON_SRCS="$NGX_ADDON_SRCS $ngx_addon_dir/ngx_http_fastdfs_module.c"
    CORE_INCS="$CORE_INCS /usr/local/include"
    CORE_LIBS="$CORE_LIBS -lfastcommon -lfdfsclient"
```

```
    CFLAGS="$CFLAGS -D_FILE_OFFSET_BITS=64 -DFDFS_OUTPUT_CHUNK_SIZE='256*1024' -
DFDFS_MOD_CONF_FILENAME='\"/etc/fdfs/mod_fastdfs.conf\"'"
fi
```

- **其中第6行和第15行要进行修改，修改后的内容如下：**

```
ngx_addon_name=ngx_http_fastdfs_module

if test -n "${ngx_module_link}"; then
    ngx_module_type=HTTP
    ngx_module_name=$ngx_addon_name
    ngx_module_incs="/usr/include/fastdfs /usr/include/fastcommon/"
    ngx_module_libs="-lfastcommon -lfdfsclient"
    ngx_module_srcs="$ngx_addon_dir/ngx_http_fastdfs_module.c"
    ngx_module_deps=
    CFLAGS="$CFLAGS -D_FILE_OFFSET_BITS=64 -DFDFS_OUTPUT_CHUNK_SIZE='256*1024' -
DFDFS_MOD_CONF_FILENAME='\"/etc/fdfs/mod_fastdfs.conf\"'"
    . auto/module
else
    HTTP_MODULES="$HTTP_MODULES ngx_http_fastdfs_module"
    NGX_ADDON_SRCS="$NGX_ADDON_SRCS $ngx_addon_dir/ngx_http_fastdfs_module.c"
    CORE_INCS="$CORE_INCS /usr/include/fastdfs /usr/include/fastcommon/"
    CORE_LIBS="$CORE_LIBS -lfastcommon -lfdfsclient"
    CFLAGS="$CFLAGS -D_FILE_OFFSET_BITS=64 -DFDFS_OUTPUT_CHUNK_SIZE='256*1024' -
DFDFS_MOD_CONF_FILENAME='\"/etc/fdfs/mod_fastdfs.conf\"'"
fi
```

拷贝mod_fastdfs.conf

将fastdfs-nginx-module-1.20/src/mod_fastdfs.conf拷贝至/etc/fdfs/下

```
cp mod_fastdfs.conf /etc/fdfs/
```

修改mod_fastdfs.conf

```
vim /etc/fdfs/mod_fastdfs.conf
```

文件内容如下：

```
base_path=/kkb/server/fastdfs/storage
tracker_Server=192.168.10.135:22122 #url中是否包含group名称
url_have_group_name=true    #指定文件存储路径，访问时使用该路径
store_path0=/kkb/server/fastdfs/storage
```

拷贝libfdfsclient.so(新版不需要)

将libfdfsclient.so拷贝至/usr/lib下

```
cp /usr/lib64/libfdfsclient.so    /usr/lib/
```

五、安装Nginx（Apache）

每一台storage server都需要安装Nginx。

下载文件

- 查看GitHub上面最新的nginx release 版本

  https://github.com/nginx/nginx/releases
- 下载各个版本的nginx的地址：

  http://nginx.org/download/
- 下载Nginx

  ```
  wget http://nginx.org/download/nginx-1.15.6.tar.gz
  ```

- 上传nginx压缩包

  ```
  yum install -y lrzsz
  rz
  ```

  通过rz命令上传文件：nginx-1.15.6.tar.gz

安装第三方软件

安装PCRE

PCRE(Perl Compatible Regular Expressions)是一个Perl库，包括 perl 兼容的正则表达式库。Nginx的http模块使用pcre来解析正则表达式，所以需要在linux上安装pcre库。

```
yum install -y pcre-devel
```

注：pcre-devel是使用pcre开发的一个二次开发库。Nginx也需要此库。

安装ZLIB

zlib库提供了很多种压缩和解压缩的方式，Nginx使用zlib对http包的内容进行gzip，所以需要在linux上安装zlib库。

```
yum install -y zlib-devel
```

安装OPENSSL

OpenSSL 是一个强大的安全套接字层密码库，囊括主要的密码算法、常用的密钥和证书封装管理功能及SSL协议，并提供丰富的应用程序供测试或其它目的使用。

Nginx不仅支持http协议，还支持https（即在ssl协议上传输http），所以需要在linux安装openssl库。

```
 yum install -y openssl-devel
```

解压缩

```
tar -xf nginx-1.15.6.tar.gz
```

执行configure配置

```
cd nginx-1.15.6/

./configure \
--prefix=/kkb/server/nginx \
--pid-path=/var/run/nginx/nginx.pid \
--lock-path=/var/lock/nginx.lock \
--error-log-path=/var/log/nginx/error.log \
--http-log-path=/var/log/nginx/access.log \
--http-client-body-temp-path=/var/temp/nginx/client \
--http-proxy-temp-path=/var/temp/nginx/proxy \
--http-fastcgi-temp-path=/var/temp/nginx/fastcgi \
--http-uwsgi-temp-path=/var/temp/nginx/uwsgi \
--http-scgi-temp-path=/var/temp/nginx/scgi \
--with-http_gzip_static_module \
--add-module=/kkb/soft/fastdfs-nginx-module-1.20/src
```

注意：

prefix=/kkb/server/nginx 中的/kkb/server/nginx指的是要安装的nginx的路径

add-module=/opt/fastdfs-nginx-module/src中的路径指的是fastdfs-nginx-module模块的解压缩路径

创建nginx/client目录

```
mkdir -p /var/temp/nginx/client
```

创建临时目录

上面执行的configure命令，设置了一些配置参数，其中的一些参数指定的目录一定要存在。

```
mkdir /var/temp/nginx -p
```

编译安装

```
make && make install
```

如果出现以下错误，请修改fastdfs-nginx-module-1.20/src/config配置文件

```
       /kkb/server/fastdfs-nginx-module-1.20/src/ngx_http_fastdfs_module.c
In file included from /kkb/server/fastdfs-nginx-module-1.20/src/common.c:26:0,
                 from /kkb/server/fastdfs-nginx-module-1.20/src/ngx_http_fastdfs_module.c:6:
/usr/include/fastdfs/fdfs_define.h:15:27: fatal error: common_define.h: No such file or directory
 #include "common_define.h"
                           ^
compilation terminated.
make[1]: *** [objs/addon/src/ngx_http_fastdfs_module.o] Error 1
make[1]: Leaving directory `/kkb/soft/nginx-1.15.6'
make: *** [build] Error 2
```

修改nginx.conf

```
vim /kkb/server/nginx/conf/nginx.conf
```

修改内容如下：

```
server {
        listen       80;
        server_name  localhost;
        location /group1/M00/{
                ngx_fastdfs_module;
        }
}
```

说明：

`location /group1/M00/`：以**/group1/M00/**开头的请求，才会正常使用Nginx模块**ngx_fastdfs_module**下载访问图片。

启动Nginx

切换到nginx/bin目录

```
./nginx
```

六、Nginx附加资料

location配置详解

语法规则：

```
location [=|~|~*|^~]  /uri/  { … }
```

语法说明：

- = 开头表示精确匹配
- ^~ 开头表示uri以某个常规字符串开头，理解为匹配 url路径即可。nginx不对url做编码，因此请求为/static/20%/aa，可以被规则^~ /static/ /aa匹配到（注意是空格）。
- **~ 开头表示区分大小写的正则匹配**
- **~* 开头表示不区分大小写的正则匹配**
- **!~和!~*分别为区分大小写不匹配及不区分大小写不匹配 的正则**
- / 通用匹配，任何请求都会匹配到。

多个location配置的情况下匹配顺序:

- 首先匹配 =
- 其次匹配^~
- 其次是按文件中顺序的正则匹配
- 最后是交给 / 通用匹配
- 当有匹配成功时候，停止匹配，按当前匹配规则处理请求。

例子，有如下匹配规则:

```
location = / {
   #规则A
}
location = /login {
   #规则B
}
location ^~ /static/ {
   #规则C
}
location ~ \.(gif|jpg|png|js|css)$ {
   #规则D
}
location ~* \.png$ {
   #规则E
}
location !~ \.xhtml$ {
   #规则F
}
location !~* \.xhtml$ {
   #规则G
}
location / {
   #规则H
}
```

那么产生的效果如下:

访问根目录/， 比如http://localhost/ 将匹配规则A

访问 http://localhost/login 将匹配规则B，http://localhost/register 则匹配规则H

访问 http://localhost/static/a.html 将匹配规则C

访问 http://localhost/a.gif，http://localhost/b.jpg 将匹配规则D和规则E，但是规则D顺序优先，规则E不起作用，而 http://localhost/static/c.png 则优先匹配到 规则C

访问 http://localhost/a.PNG 则匹配规则E， 而不会匹配规则D，因为规则E不区分大小写。

访问 http://localhost/a.xhtml 不会匹配规则F和规则G，http://localhost/a.XHTML不会匹配规则G，因为不区分大小写。规则F，规则G属于排除法，符合匹配规则但是不会匹配到，所以想想看实际应用中哪里会用到。

访问 http://localhost/category/id/1111 则最终匹配到规则H，因为以上规则都不匹配，这个时候应该是nginx转发请求给后端应用服务器，比如FastCGI（php），tomcat（jsp），nginx作为方向代理服务器存在。

所以实际使用中，通常至少有三个匹配规则定义，如下：

```
#直接匹配网站根，通过域名访问网站首页比较频繁，使用这个会加速处理，官网如是说。
#这里是直接转发给后端应用服务器了，也可以是一个静态首页
# 第一个必选规则
location = / {
    proxy_pass http://tomcat:8080/index
}

# 第二个必选规则是处理静态文件请求，这是nginx作为http服务器的强项
# 有两种配置模式，目录匹配或后缀匹配,任选其一或搭配使用
location ^~ /static/ {
    root /webroot/static/;
}
location ~* \.(gif|jpg|jpeg|png|css|js|ico)$ {
    root /webroot/res/;
}

#第三个规则就是通用规则，用来转发动态请求到后端应用服务器
#非静态文件请求就默认是动态请求，自己根据实际把握
#毕竟目前的一些框架的流行，带.php,.jsp后缀的情况很少了
location / {
    proxy_pass http://tomcat:8080/
}
```

rewrite语法

- **语法命令：**

```
rewrite     <regex>     <replacement>     [flag]

关键字          正则          替代内容           flag标记
```

- **命令解释：**
  - 关键字：重写语法关键字
  - 正则：perl兼容正则表达式语句进行规则匹配
  - 替代内容：将正则匹配的内容替换成replacement
  - flag标记：rewrite支持的flag标记
- **flag标记说明：**

```
* last   #本条规则匹配完成后，继续向下匹配新的location URI规则

* break   #本条规则匹配完成即终止，不再匹配后面的任何规则

* redirect   #返回302临时重定向，浏览器地址会显示跳转后的URL地址

* permanent   #返回301永久重定向，浏览器地址栏会显示跳转后的URL地址
```