1、分布式系统架构

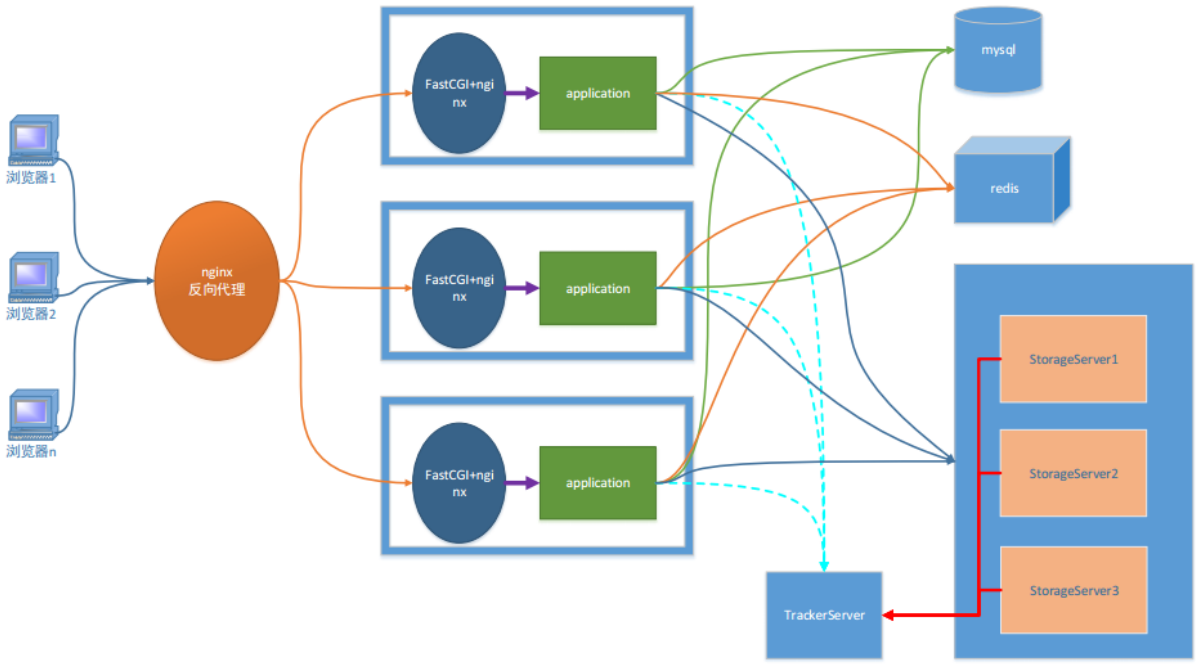
1.1 相关概念

- 服务器

  服务器在硬件上表示为一台高配置的电脑；软件上，电脑必须有一个能够解析http协议的软件。

- 常见的Web服务器
  - tomcat服务器
  - weblogic服务器
  - IIS服务器
  - nginx：小巧高效的HTTP服务器；可以作为高效的负载均衡反向代理；邮件服务器（pop3/smtp/imap）

1.2 分布式系统架构




```
1、客户端
    网络架构
        B/S --> 必须使用http协议
        C/S --> 协议可以随意选择
2、反向代理服务器（Nginx）
    客户端不会直接访问web服务器，直接访问到的是反向代理服务器
    客户端将请求发送给反向代理服务器，反向代理服务器将客户端请求转发给web服务器
    反向代理服务器是分布式的关键，通过负载均衡，会将请求发送给某一台服务器进行业务处理
3、服务器
    * Nginx
        处理静态请求 --> .html .jpg ...
        服务器集群之后，每台服务器上部署的内容必须相同
        动态请求无法处理
```

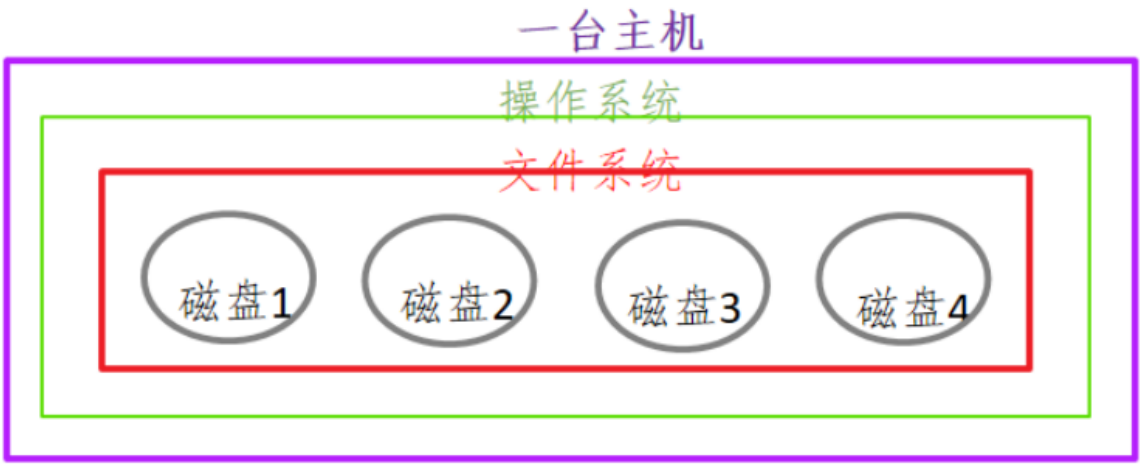
```
    * fastCGI
        帮助Nginx服务器处理动态请求，动态处理程序有程序员编写
        和Nginx部署在同一台电脑上，对Nginx补充作用
4、关系型数据库
    存储需要持久化存储的信息
    比如文件上传需要存储文件属性信息、用户注册也需要对用户数据进行存储 ...
5、非关系型数据库（redis内存数据库）
    提高程序效率
    存储的是服务器经常要从关系型数据库中读取的数据
6、fastDFS分布式文件系统
    存储文件内容
    供用户下载
```

1.3 服务器学习过程

```
分布式文件系统fastDFS --> NGINX --> FastCGI
```

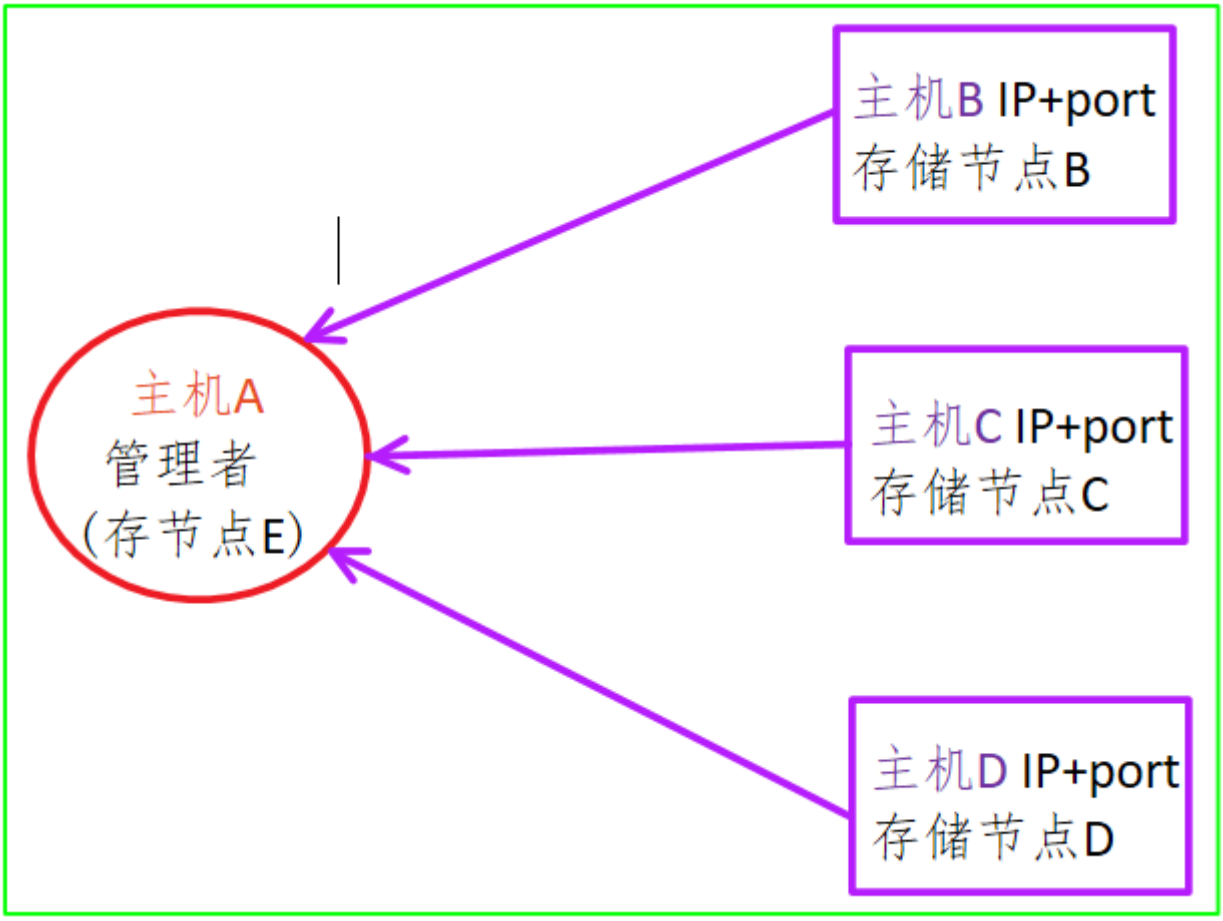
2、FastDFS分布式文件系统

2.1 传统文件系统



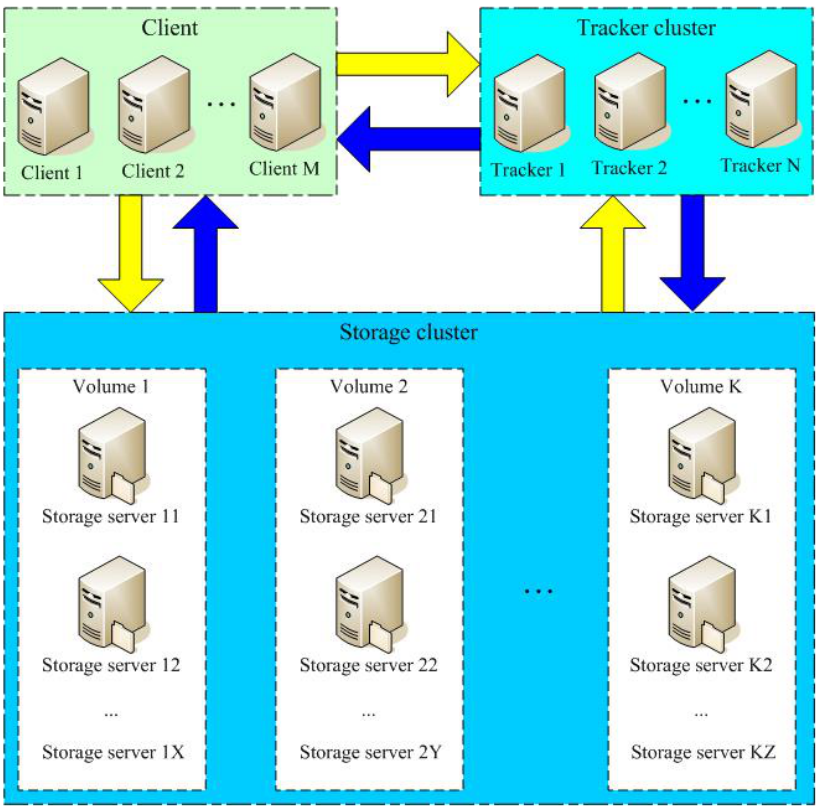
2.2 分布式文件系统

```
文件系统的全部不在同一台主机上，而是在很多主机上，多个分散的文件系统组合在一起，形成一个完整的文件系统。
分布式文件系统：
    1. 需要有网络
    2. 多台主机
        不需要在同一地点
    3. 需要管理者
    4. 编写应用层的管理程序
        不需要编写
```



2.3 FastDFS文件系统

官方的一张系统结构图：

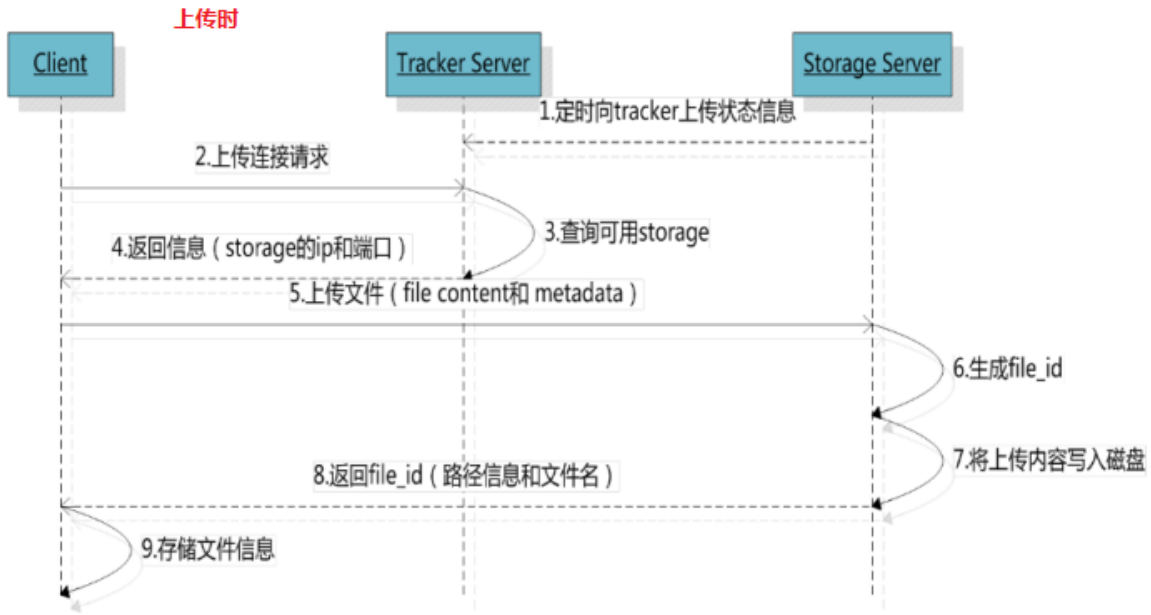


1、FastDFS概述
    * 是用c语言编写的一款开源的分布式文件系统。
    * 为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，注重高可用、高性能等指标
        冗余备份：纵向扩容
        线性扩容：横向扩容
    * 可以很容易搭建一套高性能的文件服务器集群提供文件 -->上传、下载 <-- 等服务。
        图床、网盘等

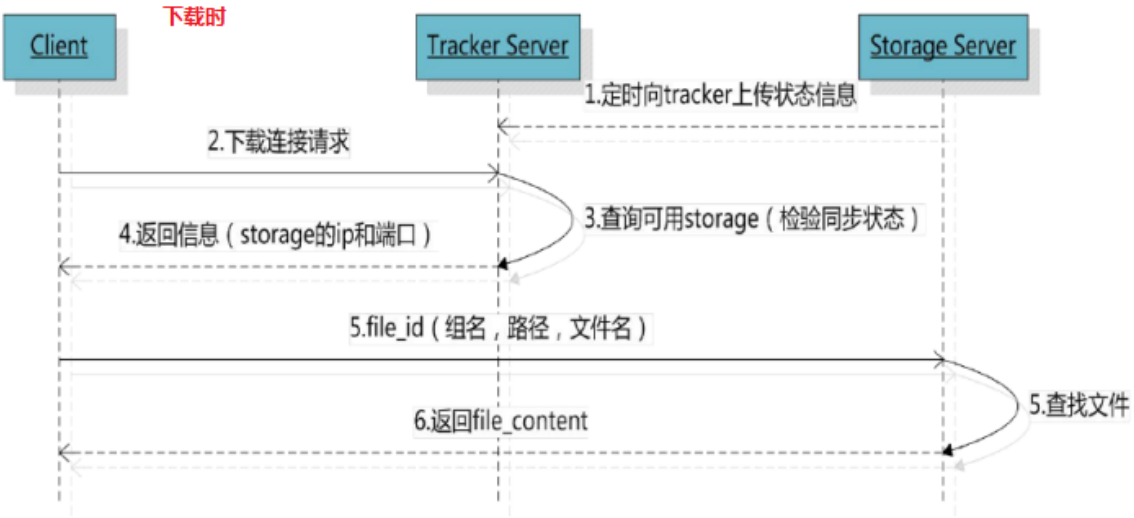
2、FastDFS中的三个角色
    * 追踪器（Tracker）- 管理者 - 守护进程
    * 管理存储节点
        存储节点 - storage - 守护进程
        存储节点是有多个的
    * 客户端 - 不是守护进程，这是程序员编写的程序
        文件上传
        文件下载

2.4 FastDFS中三个角色的关系

- 上传



- 下载



1. 追踪器
   最先启动追踪器
2. 存储节点
   第二个启动的角色
   存储节点启动之后，会单独开一个线程，向Tracker追踪器汇报一些信息
      汇报当前存储节点的容量，和剩余容量
      汇报数据的同步情况
      汇报数据被下载的次数
3. 客户端

最后启动

   上传
      连接追踪器，询问存储节点的信息
      假设要上传1G的文件，询问那个存储节点有足够的容量
      追踪器查询，得到结果
      追踪器将查到的存储节点的IP+端口发送给客户端
      通过得到IP和端口连接存储节点
      将文件内容发送给存储节点，并且返回一个该文件的唯一file_ID
   下载
      连接追踪器，传入file_ID，询问存储节点的信息
      问一下，要下载的文件在哪一个存储节点
      追踪器查询，得到结果存储节点的ip和端口
      追踪器将查到的存储节点的IP+端口发送给客户端
      通过得到IP和端口连接存储节点
      下载文件

2.5 FastDFS的安装

- 源码安装

```
1、libfastcommon-1.36.zip
    fastdfs的基础库包，fastdfs对这个包有依赖
    解压：unzip xxx.zip
    ./make.sh
    ./make.sh install
2、fastdfs-5.10.tar.gz
    解压：tar zxvf xxx.tar.gz
    ./make.sh
    ./make.sh install
```

- 安装之后各个条目所在目录

```
配置文件：
    * 配置文件是主要的部分，在启动、关闭、重启三个角色的程序时都需要在命令中指定对应的配置文件
    * 所在路径：/etc/fdfs -->
        client.conf.sample | storage.conf.sample | storage_ids.conf.sample |
tracker.conf.sample
可执行程序：
    在三个角色启动时都需要对应的可执行程序，在/usr/bin/目录中，有：
    fdfs_trackerd \ fdfs_storaged \ fdfs_upload_file \ file_download_file \
file_monitor等等
```

2.6 FastDFS启动

- 启动程序在 /usr/bin/fdfs_*

```
1、追踪器：第一个启动，是一个守护进程

# 启动
fdfs_trackerd 追踪器的配置文件(/etc/fdfs/tracker.conf)
# 关闭
fdfs_trackerd 追踪器的配置文件(/etc/fdfs/tracker.conf) stop
# 重启
fdfs_trackerd 追踪器的配置文件(/etc/fdfs/tracker.conf) restart
```

```
2、存储节点：守护进程

# 启动
fdfs_storaged 存储节点的配置文件(/etc/fdfs/stroga.conf)
# 关闭
fdfs_storaged 存储节点的配置文件(/etc/fdfs/stroga.conf) stop
# 重启
fdfs_storaged 存储节点的配置文件(/etc/fdfs/stroga.conf) restart
```

```
3、客户端上传下载操作：普通进程，用完即死

# 上传
fdfs_upload_file 客户端的配置文件(/etc/fdfs/client.conf) 要上传的文件
# 得到的结果字符串file_ID: group1/M00/00/00/wKj3h1vC-PuAJ09iAAAHT1YnUNE31352.c

# 下载
fdfs_download_file 客户端的配置文件(/etc/fdfs/client.conf) 上传成功之后得到的字符串
(fileID)
```

```
4、状态检测

fdfs_monitor （需要检测哪个，就写它的配置文件）/etc/fdfs/client.conf

7种状态：
# FDFS_STORAGE_STATUS：INIT ：初始化，尚未得到同步已有数据的源服务器
# FDFS_STORAGE_STATUS：WAIT_SYNC ：等待同步，已得到同步已有数据的源服务器
# FDFS_STORAGE_STATUS：SYNCING ：同步中
# FDFS_STORAGE_STATUS：DELETED ：已删除，该服务器从本组中摘除
# FDFS_STORAGE_STATUS：OFFLINE ：离线
# FDFS_STORAGE_STATUS：ONLINE ：在线，尚不能提供服务
# FDFS_STORAGE_STATUS：ACTIVE ：在线，可以提供服务
```

2.7 FastDFS修改配置文件

配置文件都是需要修改之后才可以使用的，使用FastDFS更多在于修改配置文件。

修改之前，将存在的client.conf.sample | storage.conf.sample | tracker.conf.sample这三个配置文件拷贝一份新的，文件名就是去掉.sample后的。

```
1、修改tracker.conf

# 将追踪器和部署的主机的IP地址进程绑定，也可以不指定
# 如果不指定，会自动绑定当前主机IP，如果是云服务器建议不要写
bind_addr=192.168.47.131    #指定
# bind_addr=                #不指定

# 追踪器监听的端口
port=22122

# 追踪器存储日志信息的目录，xxx.pid文件，必须是一个存在的目录。追踪器的日志信息都会放在这个目录中
base_path=/home/yuqing/fastdfs/tracker

###
当追踪器初始启动时，会在base_path指定的路径中创建对应的日志文件。
```

```
2、修改storage.conf
```

```
# 当前存储节点对应的主机属于哪一个组
group_name=group1

# 当前存储节点和所应该的主机进行IP地址的绑定，如果不写，有fastdfs自动绑定
bind_addr=

# 存储节点绑定的端口
port=23000

# 存储节点写log日志的路径
base_path=/home/yuqing/fastdfs/storage

# 存储节点提供的存储文件的路径个数，对应下面的store_pathx属性
store_path_count=3

# 具体的存储路径，可以有多个，当第一次启动存储节点之后就会对这三个目录进行初始化，创建很多目录，会在file_ID种体现出来
store_path0=/home/yuqing/fastdfs/loc_1
store_path1=/home/yuqing/fastdfs/loc_2
store_path1=/home/yuqing/fastdfs/loc_3

# 追踪器的地址信息，需要和上面的追踪器设置的ip和端口匹配
tracker_server=192.168.47.133:22122
tracker_server=192.168.47.131:22122  # 集群
tracker_server=192.168.47.132:22122  # 集群

####
启动之后，存储节点会在base_path指定的目录生成对应的日志信息；
        在store_pathx指定的目录中生成多个目录，用于之后的文件存储
        还会创建单独的线程，向tracker_server指定的ip和端口的追踪器汇报自身的相关信息
        存储节点角色主要有两个目录，一个是指定的日志存储目录，一个是文件存放目录，可以在初始化启动之后查看其结构
```

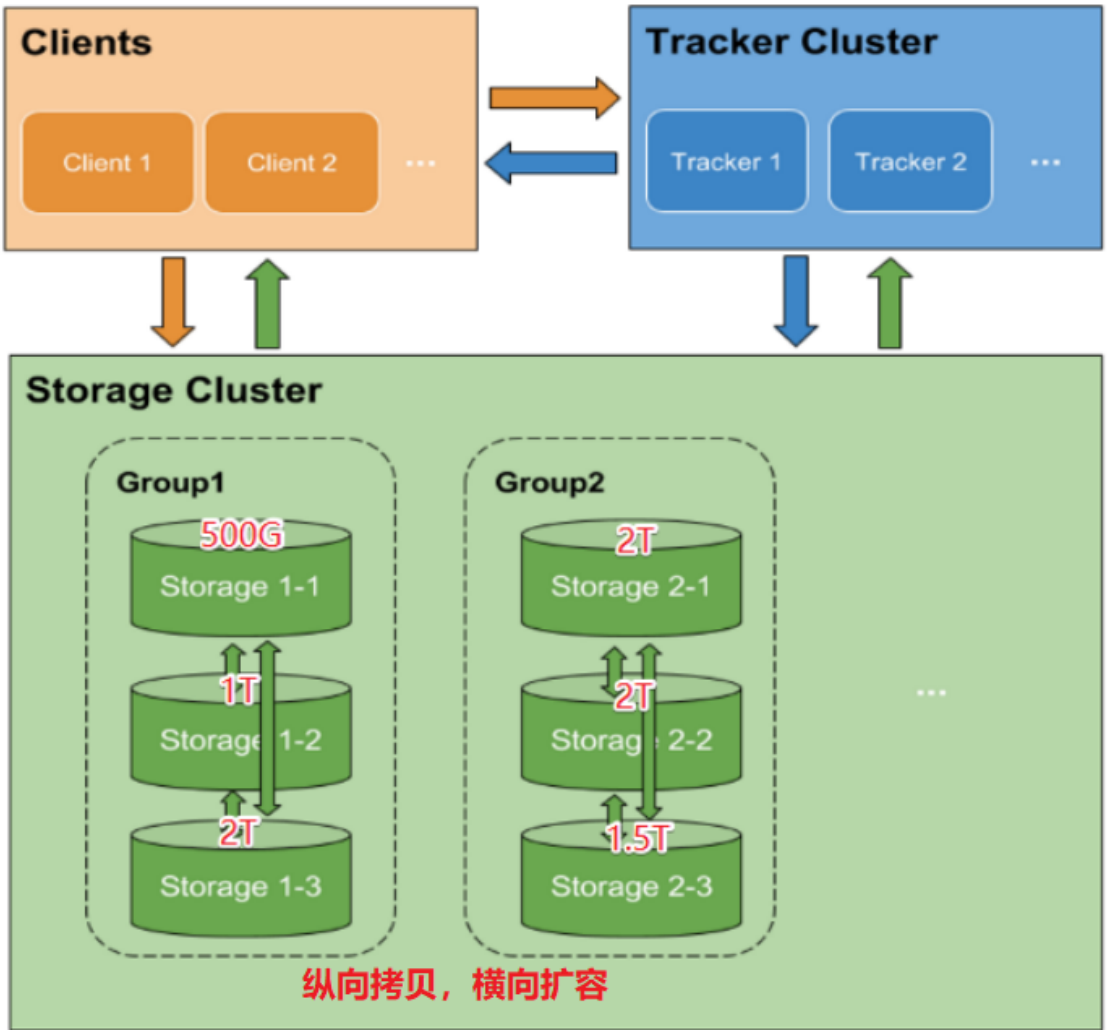
```
3、修改 client.conf
# 客户端写log日志的目录
# 该路径必须存在
# 当前的用户对于该路径中的文件有读写权限
base_path=/home/yuqing/fastdfs/client

# 要连接的追踪器的地址信息,要和追踪器的配置文件对应
tracker_server=192.168.47.133:22122
tracker_server=192.168.47.131:22122  # 集群
tracker_server=192.168.47.132:22122  # 集群
```

2.8 file_ID的解析

```
group1 - 存储节点所在的组别，如果没有集群通常就是group1
    文件上传到了存储节点的哪一个组
    如果有多个组这个组名可变的
M00 - 虚拟目录，和存储节点的配置项有映射
    store_path0=/home/yuqing/fastdfs/loc_1/data  -> M00
    store_path1=/home/yuqing/fastdfs/loc_2/data  -> M01
    ...
00/00 - 实际的路径，可变的。这个路径是在初始化存储节点是创建的，
在/home/yuqing/fastdfs/loc_1/data中

wKhS_VlrEfOAdIZyAAAJTOwCGr43848.c - 文件名包含的信息及编码
采用Base64编码
    包含的字段包括
        源storage server Ip 地址
        文件创建时间
        文件大小
        文件CRC32效验码
            循环冗余校验
        随机数
```

2.9 FastDFS分布式集群

上面的配置中，在存储节点和客户端的tracker_server中配置了三个追踪器，这就是配置了一个集群的FastDFS系统。在最开始的官方图片中，就是分布式集群配置之后的架构图。亦可参考下图：



纵向拷贝问题：
写文件时，客户端将文件写至group内一个storage server即认为写文件成功，storage server写完文件后，会由后台线程将文件同步至同group内其他的storage server。

文件同步(添加/删除/修改)storage server之间进行，采用push方式，即源服务器同步给目标服务器；

每个storage写文件后，同时会写一份binlog，binlog里不包含文件数据，只包含文件名等元信息，这份binlog用于后台同步，storage会记录向group内其他storage同步的进度，以便重启后能接上次的进度继续同步；进度以时间戳的方式进行记录，所以最好能保证集群内所有server的时钟保持同步。

storage的同步进度会作为元数据的一部分汇报到tracker上，tracke在选择读storage的时候会以同步进度作为参考。

源头数据才需要同步，备份数据不需要再次同步，否则就会构成环路了

集群：

1、追踪器集群
为什么集群？
　　避免单点故障，一但一个追踪器崩了，可以有其他追踪器工作。
多个Tracker如何工作？
　　轮询工作

如何实现集群？
只要有多个不同IP和端口的追踪器即可，追踪器之间不需要配置文件配置任何关系。

2、存储节点集群
fastDFS管理存储节点的方式？
通过分组的方式完成的
集群方式（扩容方式）
* 横向扩容 - 增加容量
添加一台新的主机 -> 容量增加了
假设当前有两个组: group1, group2
需要添加一个新的分组 -> group3 新主机属于第三组
不同组的主机之间不需要通信
* 纵向扩容 - 数据备份
假设当前有两个组: group1, group2
将新的主机放到现有的组中
每个组的主机数量从1 -> N
这n台主机的关系就是相互备份的关系
同一个组中的主机需要通信
每组的容量 == 容量最小的这台主机
如何实现?
同追踪器，只要有多个指定了不同ip和端口的配置文件，然后通过命令启动即可，甚至在一台主机上都可以配置多个存储节点实现集群。

3、客户端
不需要配置多少个，因为不同的用户就是一个集群。只是在客户端配置文件中要将tracker_server配置多个。

2.10 代码实现上传下载

```
//##############################################################################
// 上传文件代码
//##############################################################################
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include "fdfs_client.h"
/**
* @brief 使用框架提供的方式上传文件并返回文件ID
*
* @param confFile 客户端配置文件路径
* @param uploadfile 需要上传的文件所在本地的路径
* @param fileID 返回参数，文件在服务器中的ID
* @return int 执行成功与否
*/
```

```c
int upload_file_1(const char *confFile, const char *uploadfile, char *fileID)
{
    char group_name[FDFS_GROUP_NAME_MAX_LEN + 1];
    ConnectionInfo *pTrackerServer;
    int result;
    int store_path_index;
    ConnectionInfo storageServer;
    if ((result = fdfs_client_init(confFile)) != 0)
    {
        return result;
    }
    pTrackerServer = tracker_get_connection();
    if (pTrackerServer == NULL)
    {
        fdfs_client_destroy();
        return errno != 0 ? errno : ECONNREFUSED;
    }
    *group_name = '\0';
    if ((result = tracker_query_storage_store(pTrackerServer,
        &storageServer, group_name,
        &store_path_index)) != 0)
    {
        fdfs_client_destroy();
        fprintf(stderr, "tracker_query_storage fail, "
            "error no: %d, error info: %s\n",
            result, STRERROR(result));
        return result;
    }
    result = storage_upload_by_filename1(pTrackerServer,
        &storageServer, store_path_index,
        uploadfile, NULL,
        NULL, 0, group_name, fileID);
    if (result == 0)
    {
        printf("%s\n", fileID);
    }
    else
    {
        fprintf(stderr, "upload file fail, "
            "error no: %d, error info: %s\n",
            result, STRERROR(result));
    }
    tracker_disconnect_server_ex(pTrackerServer, true);
    fdfs_client_destroy();
    return result;
}


/**
* @brief 使用执行命令的方式上传文件
*
* @param confFile 客户端配置文件所在路径
```

```
* @param uploadfile 上传的文件
* @param fileID 返回的文件ID
* @param size fileID的长度
*/
void upload_file_2(const char *confFile, const char *uploadfile, char *fileID, int
size)
{
    // 1、创建匿名管道
    int fd[2] = { 0 };
    int ret = pipe(fd);
    if (ret == -1)
    {
        perror("pipe error: ");
        exit(0);
    }
    // 2、创建子进程
    pid_t pid = fork();
    if (pid == 0)
    {
        // 子进程
        // 将标准输出重定向到管道写端，输出上传之后的文件ID
        dup2(fd[1], STDOUT_FILENO);
        // 关闭读端
        close(fd[0]);
        // 执行命令
        execlp("fdfs_upload_file", "fdfs_upload_file", confFile, uploadfile,
            NULL);
        perror("execlp: ");
    }
    else
    {
        // 父进程，读管道。接受上传之后的文件ID
        // 关闭写端
        close(fd[1]);
        read(fd[0], fileID, size);
        // 回收子进程
        wait(NULL);
    }
}

/**
* @brief 测试两种上传文件的方式
*
* @return int
*/
int main(void)
{
    char fileID[1024] = { 0 };
    // 1、通过框架提供的方式上传问题
    // upload_file_1("/etc/fdfs/client.conf", "fastdfs_upload_file.c", fileID);
    // printf("上传之后的文件ID: %s\n", fileID);
    //-----------------------------------------------
```

```
    // 2、通过命令方式
    memset(fileID, 0x0, sizeof fileID);
    upload_file_2("/etc/fdfs/client.conf", "fastdfs_upload_file.c", fileID,
        sizeof fileID);
    printf("上传之后的文件ID: %s\n", fileID);
}
```

```
//##############################################################################
// 下载文件代码
//##############################################################################
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "fdfs_client.h"
/**
* @brief 利用fastdfs提供的代码改写下载代码
*
* @param conf_filename 客户端配置文件路径
* @param local_file_id 上传文件时返回的ID
* @param to_savefile 下载保存到本地文件路径
* @param file_offset 下载文件时偏移的字节数
* @param download_bytes 需要下载的文件的字节数，需要准确数值
* @return int
*/
int fastdfs_download_file_1(const char *conf_filename, const char*local_file_id,
char *to_savefile,
    const int64_t file_offset, const int64_t download_bytes)
{
    // 与tracker的连接对象
    ConnectionInfo *pTrackerServer;
    // 存放各种函数的执行结果
    int result;
    char file_id[128];
    // 借助客户端配置文件初始化客户端
    if ((result = fdfs_client_init(conf_filename)) != 0)
    {
        return result;
    }
    // 获取与tracker的连接对象
    pTrackerServer = tracker_get_connection();
    if (pTrackerServer == NULL)
    {
        fdfs_client_destroy();
        return errno != 0 ? errno : ECONNREFUSED;
    }
```

```
    // 将传入的文件ID放到file_id中
    // snprintf(file_id, sizeof(file_id), "%s", local_filename);
    int64_t file_size = 0;
    result = storage_do_download_file1_ex(pTrackerServer,
        NULL, FDFS_DOWNLOAD_TO_FILE,
        local_file_id,
        file_offset, download_bytes,
        &to_savefile, NULL, &file_size);
    if (result != 0)
    {
        printf("download file fail, "
            "error no: %d, error info: %s\n",
            result, STRERROR(result));
    }
    // 断开连接，释放客户端
    tracker_disconnect_server_ex(pTrackerServer, true);
    fdfs_client_destroy();
}

int main(int argc, char *argv[])
{
    // fastdfs_download_file_1(const char *conf_filename, const char
    *local_file_id, const char *to_savefile,
        // const int64_t file_offset, const int64_t
        download_bytes);
        fastdfs_download_file_1(
            "/etc/fdfs/client.conf",
            "group1/M00/00/00/wKgvg2Qwb6mAX98WAAAOfCxQj8w13066.c",
            "./download.c",
            1027,
            2681); // 必须是文件确定的大小
        return 0;
}
```

3、Nginx高性能服务器

3.1 Nginx基本介绍

- Nginx是C语言编写的开源框架。
- Nginx可以作为web服务器，反向代理服务器和邮件服务器。
- 效率高，响应请求更快；高扩展；可靠；低内存消耗；单机支持10万以上并发连接；热部署和最自由的BSD许可协议，用户可以免费使用，修改源码再发布。

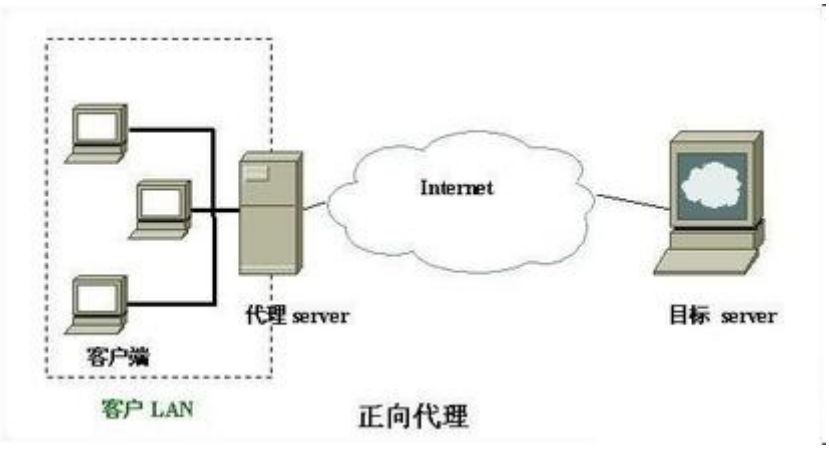
3.2 正向代理和反向代理

- 正向代理

  正向代理是位于客户端和原始服务器之间的服务器，为了能够从原始服务器获取请求的内容，客户端需要将请求发送给代理服务器，然后再由代理服务器将请求转发给原始服务器，原始服务器接受到代理服务器的请求并处理，然后将处理好的数据转发给代理服务器，之后再由代理服务器转发发给客户端，完成整个请求过程。

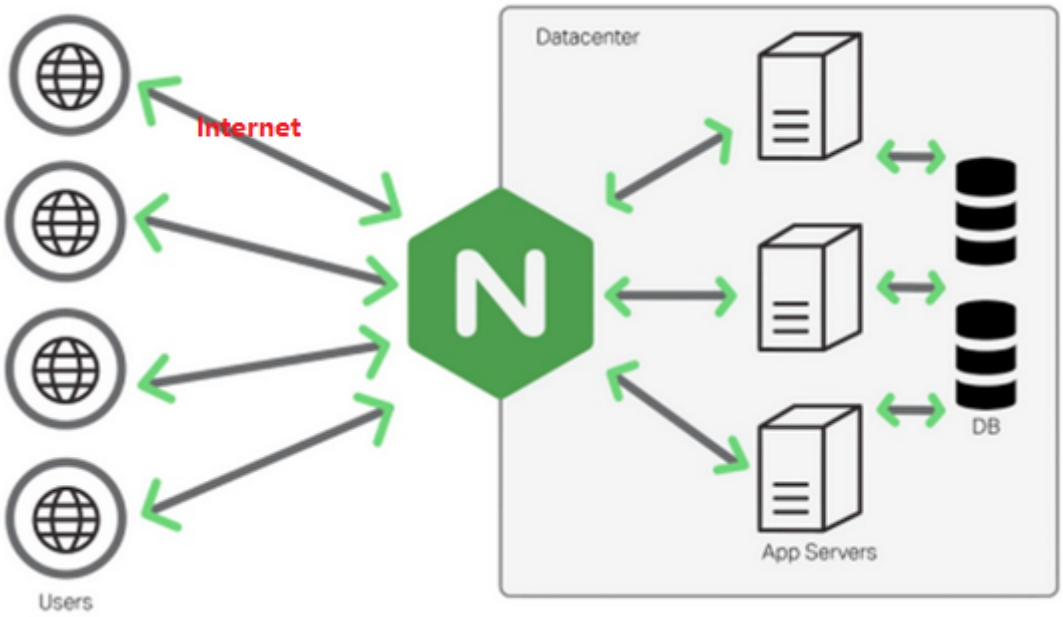
  正向代理的典型用途就是为在防火墙内的局域网客户端提供访问Internet的途径，比如：

  - 学校局域网
  - 单位局域网访问外部资源



- 反向代理

  反向代理方式是指代理原始服务器来接受来自Internet的链接请求，然后将请求转发给内部网络上的原始服务器，并将从原始服务器上得到的结果转发给Internet上请求数据的客户端。那么顾名思义，反向代理就是位于Internet和原始服务器之间的服务器，对于客户端来说就表现为一台服务器，客户端所发送的请求都是直接发送给反向代理服务器，然后由反向代理服务器统一调配。



```
1. 客户端给服务器发送请求，连接服务器，用户不知道服务器地址，只有反向代理服务器的地址是公开的
2. 请求直接发给反向代理服务器
3. 反向代理服务器将请求转发给后边的web服务器
   - web服务器 N 台
   - 反向代理服务器转发请求会轮询进行
4. web服务器收到请求进行处理，得到结果
5. web服务器将处理结果发送给反向代理服务器
6. 反向代理服务器将拿到的结果转发给客户端
```

- 两者的区别

  假设正在使用美团点外卖，在app上指定了某一家的外卖烧鸡，然后外卖员就会代替你去这家店取餐交给你，那么这相当于正向代理。

  如果美团外卖只有货物而提供店铺信息，那么你只需要点个烧鸡，那么外卖员就会替你选择一家的烧鸡去取餐交给你，那么这相当于反向代理。

3.3 域名和IP

```
1. 什么是域名？
   - www.baidu.com
   - jd.com
   - taobao.com
2. 什么是IP地址？
   - 点分十进制的字符串
     - 11.22.34.45
3. 域名和IP地址的关系？
   - 域名绑定IP
     - 一个域名只能绑定一个IP
     - 一个IP地址被多个域名绑定
```

3.4 Nginx的安装和配置

- 环境准备

```
1. 官方地址: http://nginx.org/ 可以下载Nginx安装包
2. Nginx相关依赖:
    OpenSSL: http://www.openssl.org/
        密码库
        使用https进行通信的时候使用
    ZLib下载: http://www.zlib.net/
        数据压缩
        安装:
        ./configure
        make
        sudo make install
    PCRE下载: http://www.pcre.org/
        解析正则表达式
        安装
```

```
./configure
make
sudo make install
```

Nginx安装之前需要安装以上三个依赖，分别是OpenSSL、ZLib和PCRE。

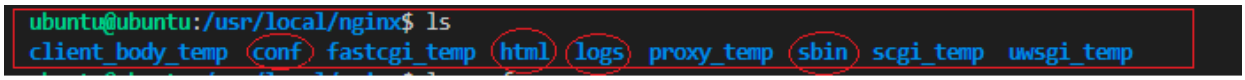
- 安装Nginx

```
从官网下载Nginx安装包，格式为xxx.tar.gz
1、解压 tar -zxvf xxx.tar.gz
2、进入解压后的目录
3、使用命令sudo ./configure --with-openssl=../openssl-1.0.1t --with-pcre=../pcre-8.40
--with-zlib=../zlib-1.2.11
4、sudo make
5、sudo make install
```

3.5 Nginx安装目录介绍

```
安装目录之后，Nginx在目录/usr/local/nginx中：
    conf -> 存储配置文件的目录
    html -> 默认的存储网站(服务器)静态资源的目录 [图片, html, js, css]
    logs -> 存储log日志
    sbin -> 启动nginx的可执行程序，这个目录中有一个程序，负责nginx的启动，重启和停止操作。
```

- 安装目录 /usr/local/nginx 分析如下



在Nginx目录中，主要有conf配置文件目录、html静态资源目录、logs日志目录和sbin可执行程序目录

- html资源目录如下

主要是两个html类型的文件　　是Nginx作为web服务器时的根目录

- logs日志目录如下

日志目录，主要是Nginx的运行日志文件，error是Nginx发生错误时的日志文件

- sbin可执行文件目录如下

有一个可执行程序nginx，负责启动Nginx，重启以及停止

- conf配置文件目录如下

```
ubuntu@ubuntu:/usr/local/nginx$ ls conf
fastcgi.conf          fastcgi_params.default  mime.types          nginx.conf.default   uwsgi_params
fastcgi.conf.default  koi-utf                 mime.types.default  scgi_params          uwsgi_params.default
fastcgi_params        koi-win                 nginx.conf          scgi_params.default  win-utf
```

nginx.conf是Nginx的核心配置文件目录，主要就是修改这个配置文件从而实现web服务，负载均衡等功能。
其他配置文件在合适的地方也有用处

3.6 Nginx的启动、重启和停止

```
1、Nginx可执行程序 /usr/local/nginx/sbin/nginx
    因为这个nginx命令不在环境变量中，所以不能直接使用，而是需要使用上面的绝对路径。
    即：sudo /usr/local/nginx/sbin/nginx # 启动
    # 快速启动方式：两种方式
    # 1、将/usr/local/nginx/sbin/nginx添加到环境变量中
    # 2、/usr/local/nginx/sbin/nginx创建软连接，放到PATH对应的目录中，比如：/usr/bin
        ln -s /usr/local/nginx/sbin/nginx /usr/bin/nginx
 启动Nginx - 需要管理员权限
 sudo nginx # 启动
```

```
2、关闭Nginx
# 第一种，马上关闭
sudo nginx -s stop
# 第二种，等nginx做完当前操作之后关闭
sudo nginx -s quit
```

```
3、重新加载Nginx
sudo nginx -s reload # 主要是修改了nginx的配置文件之后，需要执行这个命令
```

```
4、当启动Nginx之后，可以在浏览器中输入本机的网址即可访问到一个页面，这个页面来源于上面html目录中的index.html。
```

3.7 Nginx配置文件详解（核心部分）

```
Nginx 配置文件的位置：
/usr/local/nginx/conf/nginx.conf

配置文件结构如下：
--main
    --http              # http模块，http相关的通信设置
        --server         # server模块，每个server对应的是一台web服务器
            --location    # location模块，处理的是客户端的请求
     --mail              # 邮件模块，处理邮件相关动作
```

```
################################
#    内部涉及到的目录和文件相对于 /usr/local/nginx目录
#    有的指令在多个块中设置了，这些设置不是都其效果的，而是采取就近原则
```

```
##############################

#user  nobody;  # 启动之后的worker进程属于谁

# 指定Nginx工作进程的数量，通常设置为CPU核心数的两倍
worker_processes  1;

# 错误日志写入的文件
#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

# 指定pid文件，里边是nginx的进程ID
#pid        logs/nginx.pid;

events {
    # 多路复用模型，使用epoll
    use epoll

    # 指定每个工作进程可以处理的最大连接数。
    # 这个只需要根据服务器的硬件配置和负载来调整。
    worker_connections  1024;
}

# http通信模块
http {
    # 包含互联网媒体类型
    include       mime.types;
    default_type  application/octet-stream;

    # 指定输出的日志的格式
    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    # 成功访问日志
    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    # 长连接超时时间
    #keepalive_timeout  0;
    keepalive_timeout  65;

    # 启用或者禁止压缩
    #gzip  on;

    # 每个server模块可以看作一台web服务器
    server {
        # web服务器监听的端口
        listen       80;
```

```
        # 对应一个域名，客户端通过该域名访问服务器
        server_name  localhost;

        # 指定编码
        charset utf8;

        # 该server被成功访问之后的日志输出文件所在
        access_log  logs/host.access.log  main;

        # 发生404错误的页面所要返回给客户端的页面
        error_page  404              /404.html;

        # 发生服务器错误所要返回给客户端的页面
        error_page   500 502 503 504  /50x.html;

        # location负责处理客户端的请求。有多个location，负责处理来自客户端的不同指令
        # server模块作为web服务器，反向代理和负载均衡，不同角色处理动作不同。
        location / {
            # root指定访问资源根目录，在nginx安装目录中的html文件夹，资源从这个html目录中获取
            root   html;
            # index指令负责响应对应的资源
            index  index.html index.htm;
        }
    }
}
```

location配置的通配规则

nginx的匹配规则不是按照location的先后顺序选定的，而是通过匹配程度高低决定的，匹配度最高的负责处理，且不再寻找其他location。

```
# 1、= 精确匹配
location =/ {}  # 匹配到http://www.example.org/

# 2、~ 大小写敏感
location ~/Example/ {}  # http://www.example.org/Example/成功 example失败
```

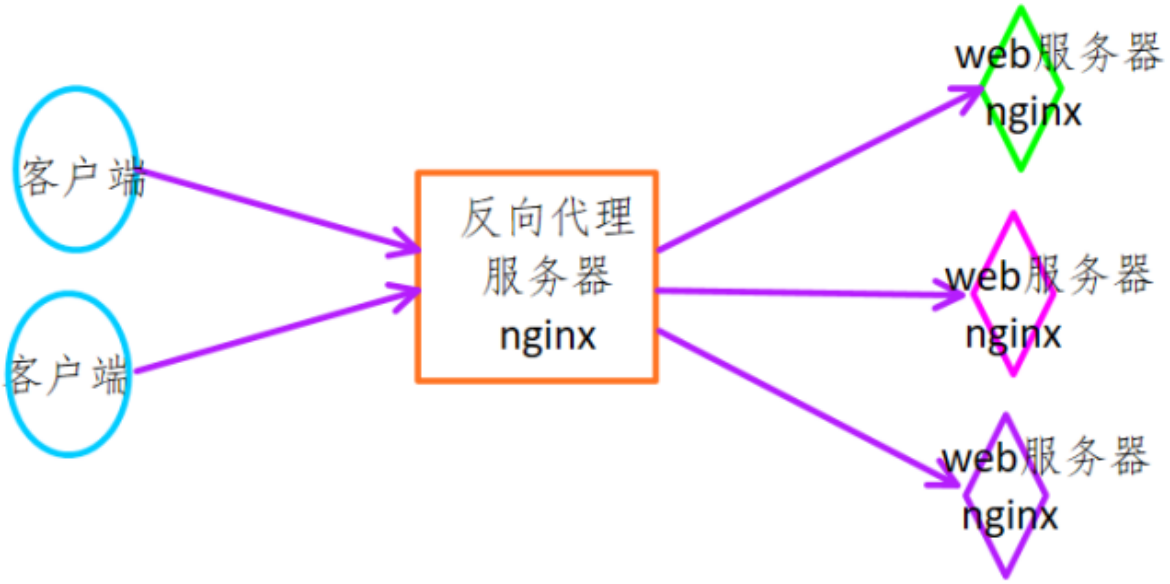
3.8 Nginx部署静态网页 - 配置文件1

Nginx可以作为web服务器，因此需要配置静态网页

```
# 请求路径：http://localhost/api/upload.html
# 其中api是值资源目录中的一个子目录，upload.html是请求的文件
# 这个api的目录必须在资源目录中存在 ***
# 相当于http://localhost/api/upload.html == http://localhost/api ==
http://localhost/api/
```

```
# 同理：http://localhost/api/api1/upload.html，其中api和api1也必须是资源文件中的目录，api1是api的子目录
# 总之，只要url中不是文本类型，都是目录，且要在服务器中的资源目录中存在。
location /api/           # 也可以不写最后的 / 即 /api
{
    root   html;
    index  upload.html index.htm;
}

location /api/api1/      # /api/api1
{
    root   html;
    index  upload.html index.htm;
}
```

3.9 Nginx部署反向代理服务器 - 配置文件2

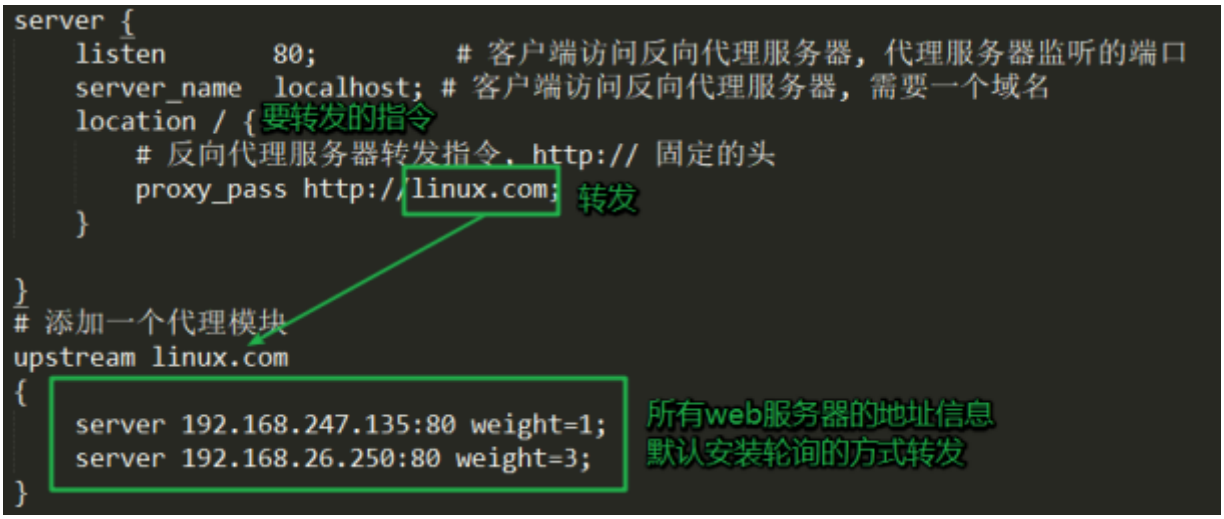



```
# 代理几台服务器就需要几个server模块
# 客户端访问的url: http://192.168.1.100/login.html
server {
    listen       80;          # 客户端访问反向代理服务器，代理服务器监听的端口
    server_name  ubuntu.com; # 客户端访问反向代理服务器，需要一个域名
    location / {  转发某一个请求
        # 反向代理服务器转发指令，http:// 固定的头
        proxy_pass http://robin.test.com;  指定转发地址，头固定，名字自己定
    }

}
# 添加一个代理模块
upstream robin.test.com
{
    server 192.168.247.135:80; web服务器的地址
}
```

反向代理是将来自客户端的请求转发到对应的web服务器中。

```
# 代理几台服务器就需要几个server模块
# 客户端访问的url: http://192.168.1.100/login.html

# robin
server
{
    listen 80;                  # 客户端访问反向代理服务器, 代理服务器监听的端口
    server_name ubuntu.com;      # 客户端访问反向代理服务器, 需要一个域名
    location /
    {
        # 反向代理服务器转发指令, http:// 固定
        proxy_pass http://robin.test.com;
    }
}
# 添加一个代理模块
upstream robin.test.com
{
    server 192.168.247.135:80;
}

# luffy
server
{
    listen 80; # 客户端访问反向代理服务器, 代理服务器监听的端口
    server_name hello.com; # 客户端访问反向代理服务器, 需要一个域名
    location /
    {
        # 反向代理服务器转发指令, http:// 固定
        proxy_pass http://luffy.test.com;
    }
}
# 添加一个代理模块
upstream luffy.test.com
{
    server 192.168.26.250:80;
}
```

3.10 Nginx负载均衡 - 配置文件3



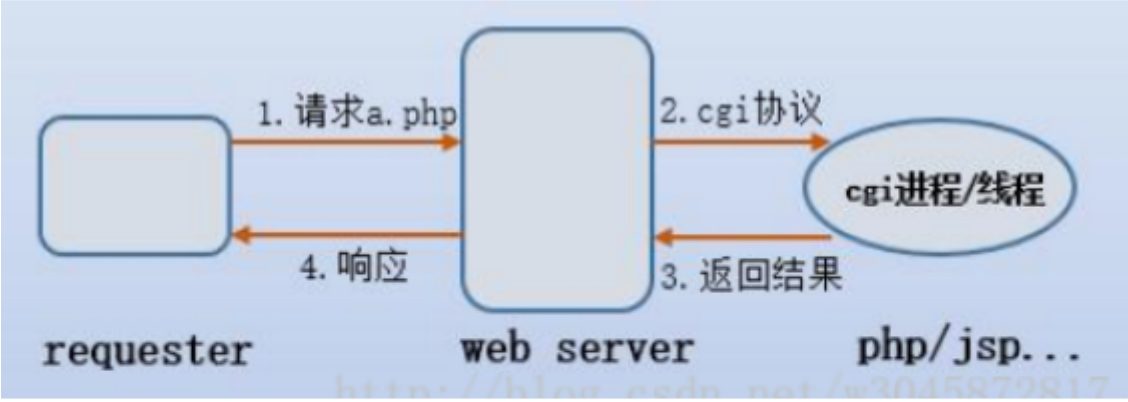

```
# 负载均衡是将用户请求交给n个服务器中的某一个，可以根据weight权重选择某一个服务器，若是不设置weight，则默认轮询
# 负载均衡场景：多个web服务器被代理服务器代理，这些web服务器部署同一个应用。location 配置只需要配置 location /    即可代理所有url，被代理的web服务器中的location写多个，负责不同的用户请求。
server
{
    listen 80; # 客户端访问反向代理服务器，代理服务器监听的端口
    server_name localhost; # 客户端访问反向代理服务器，需要一个域名

    location /
    {
        # 反向代理服务器转发指令，http:// 固定的头
        proxy_pass http://linux.com;
    }
    location /hello/ #可不在代理服务器中配置
    {
        # 反向代理服务器转发指令，http:// 固定的头
        proxy_pass http://linux.com;
    }
    location /upload/ #可不在代理服务器中配置
    {
        # 反向代理服务器转发指令，http:// 固定的头
        proxy_pass http://linux.com;
    }
}

# 添加一个代理模块
upstream linux.com
{
    server 192.168.247.135:80 weight=1;
    server 192.168.26.250:80 weight=3;
}

################ 被代理的web服务器的配置 ########################
# 192.168.247.135
location /
{
```

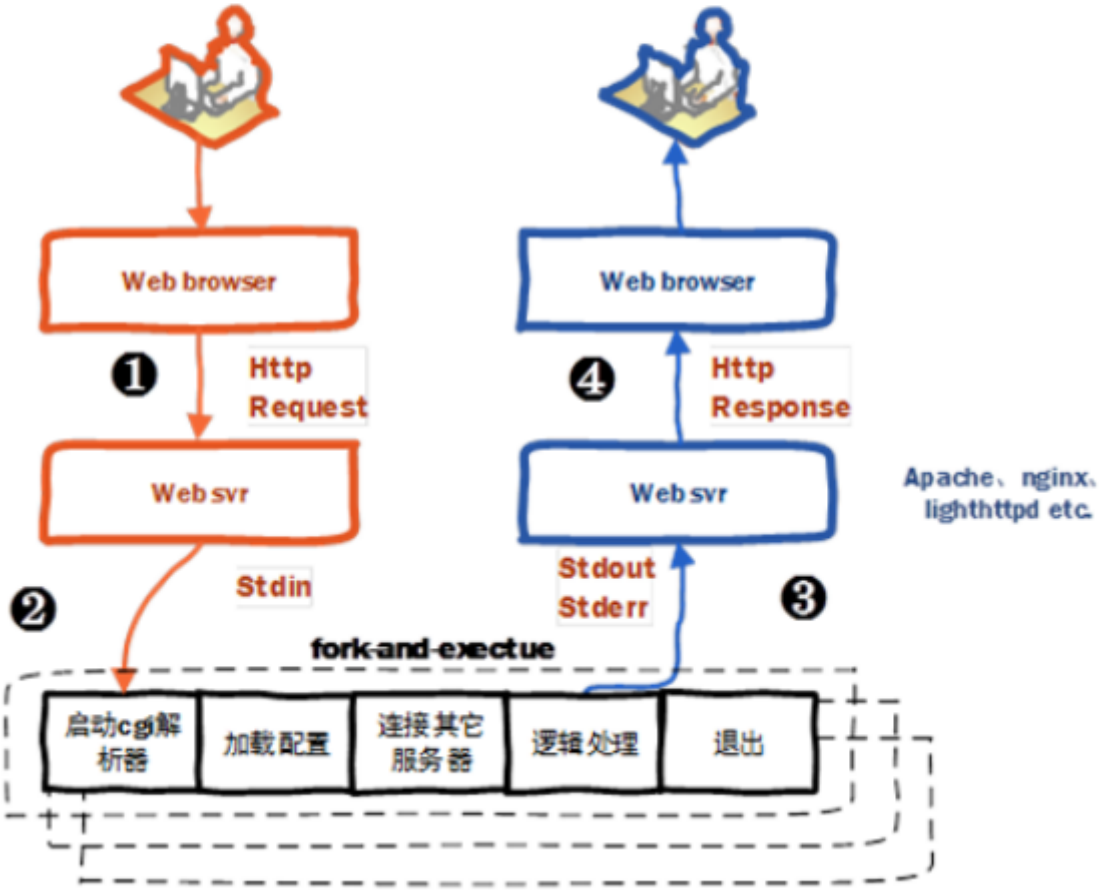
```
    root xxx;
    index xxx
}
location /hello/
{
    root xx;
    index xxx;
}
location /upload/
{
    root xxx;
    index xx;
}

# 192.168.26.250
location /
{
    root xxx;
    index xxx;
}
location /hello/
{
    root xx;
    index xxx;
}
location /upload/
{
    root xxx;
    index xx;
}
```

4、FastCGI快速通用网关接口

4.1 通用网关接口CGI

> 通用网关接口（Common Gateway Interface/CGI）描述了客户端和服务器程序之间传输数据的一种标准，可以 让一个客户端，从网页浏览器向执行在网络服务器上的程序请求数据。CGI 独立于任何语言的，CGI 程序可以用 任何脚本语言或者是完全独立编程语言实现，只要这个语言可以在这个系统上运行。





> 动态资源访问过程
>
> 请求：http://localhost/login?name=zhangsan&age=12&gender=man
>
> 1. 用户通过浏览器访问服务器, 发送了一个请求, 请求的url如上
> 2. 服务器接收数据，对接受的数据进行解析
> 3. nginx对于一些登录数据无法处理，nginx将数据发送给了CGI程序
>   - 服务器端会创建一个cgi进程进行业务处理
> 4. CGI进程执行
>   - 加载配置，如果有需求加载配置文件获取数据
>   - 连接其他服务器，比如数据库
>   - 逻辑处理

- 得到结果，将结果发送给服务器
- 退出

5. 服务器将CGI处理结果发送给客户端

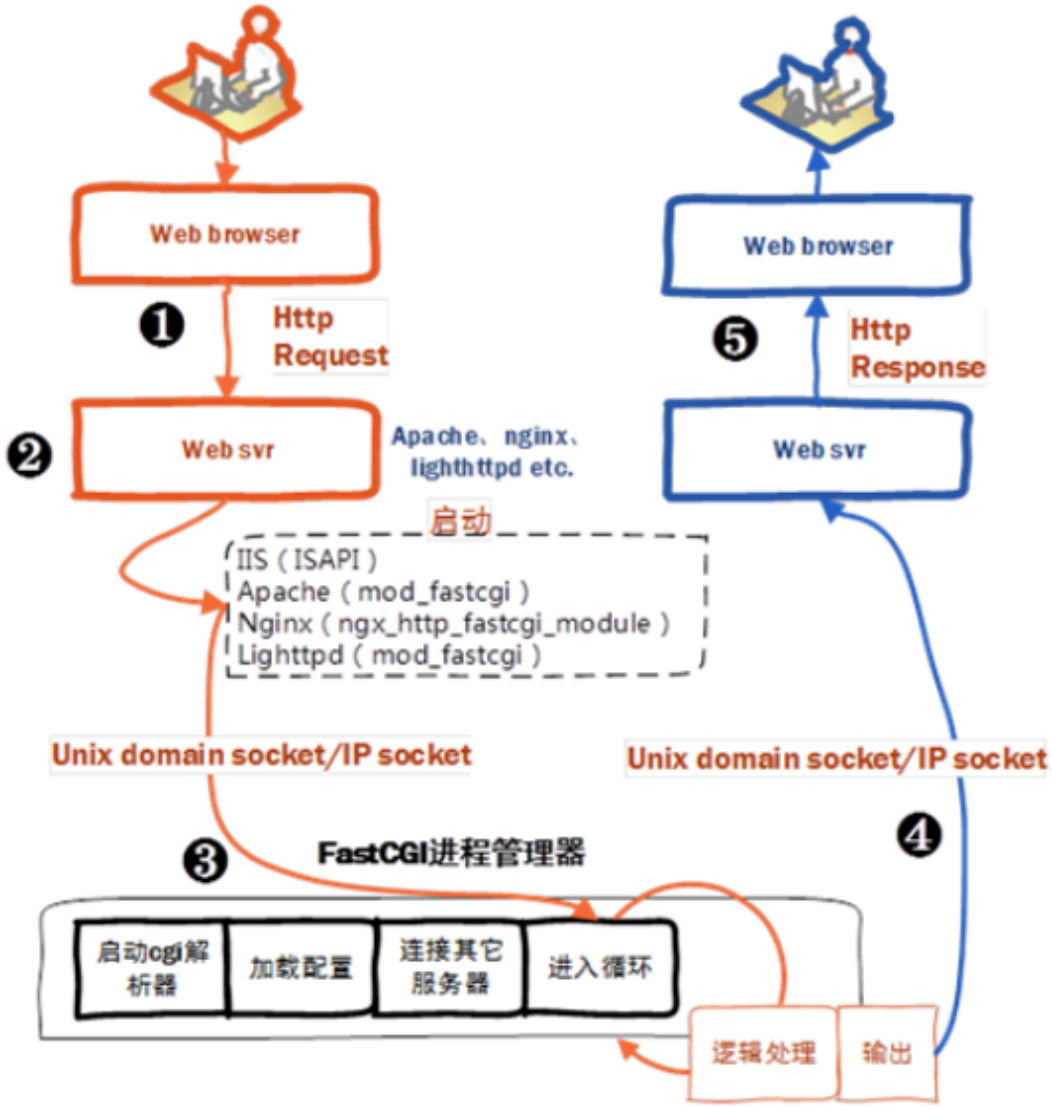
在服务器端CGI进程会被频繁的创建销毁，这会导致服务端开销很大。

4.2 FastCGI快速通用网关接口

快速通用网关接口（Fast Common Gateway Interface / FastCGI）是通用网关接口（CGI）的改进，描述了客户端 和服务器程序之间传输数据的一种标准。 FastCGI致力于减少Web服务器与 CGI 程式 之间互动的开销，从而使 服务器 可以同时处理更多的Web请求 。与为每个请求创建一个新的进程不同，FastCGI使用持续的进程来处理 一连串的请求。这些进程由FastCGI进程管理器管理，而不是web服务器。

FastCGI与CGI的区别：

CGI 就是所谓的短生存期应用程序，FastCGI 就是所谓的长生存期应用程序。FastCGI像是一个常驻(long-live)型的 CGI，它可以一直执行着，不会每次都要花费时间去fork一次



> 动态资源访问过程
>
> 请求：http://localhost/login?name=zhangsan&age=12&gender=man
>
> 1. 用户通过浏览器访问服务器，发送了一个请求
> 2. 服务器接收数据，对接收的数据进行解析
> 3. nginx对于登录数据无法处理，nginx将数据发送给fastcgi程序
>   - 通过本地套接字或者网络通信方式
> 4. fastCGI程序如何启动
>   - 不是有web服务器直接启动
>   - 而是通过一个fastCGI进程管理器启动
> 5. fastCGI启动
>   - 加载配置 - 可选
>   - 链接服务器 - 数据库
>   - 循环
>   - 服务器有请求 --> 处理

- 将处理结果发送给服务器
  - 网络通信 / 本地套接字
- 没有请求 --> 阻塞

6. 服务器将fastCGI的处理结果发送给客户端

4.3 FastCGI和spawn-fcgi安装

1、安装FastCGI，开发包

```
解压安装包，进入解压后的目录

./configure
make
make之后可能出现错误：fcgio.cpp:50:14: error: 'EOF' was not declared in this scope
                    没有包含对应的头文件：
                    stdio.h - c
                    cstdio -> c++
sudo make install
```

2、安装spawn-fcgi，CGI进程管理器

```
./configure
make
sudo make install
```

3、相关说明

```
第一个安装包fastCGI，是负责安装开发时需要使用的动态库的。
第二个安装包spawn，负责安装一个程序，作为一个服务，可以通过命令 spawn-fcgi -a IP地址 -p 端口 -f fastcgi可执行程序 直接启动。
```

fastCGI安装的动态库相关条目安装路径如下：

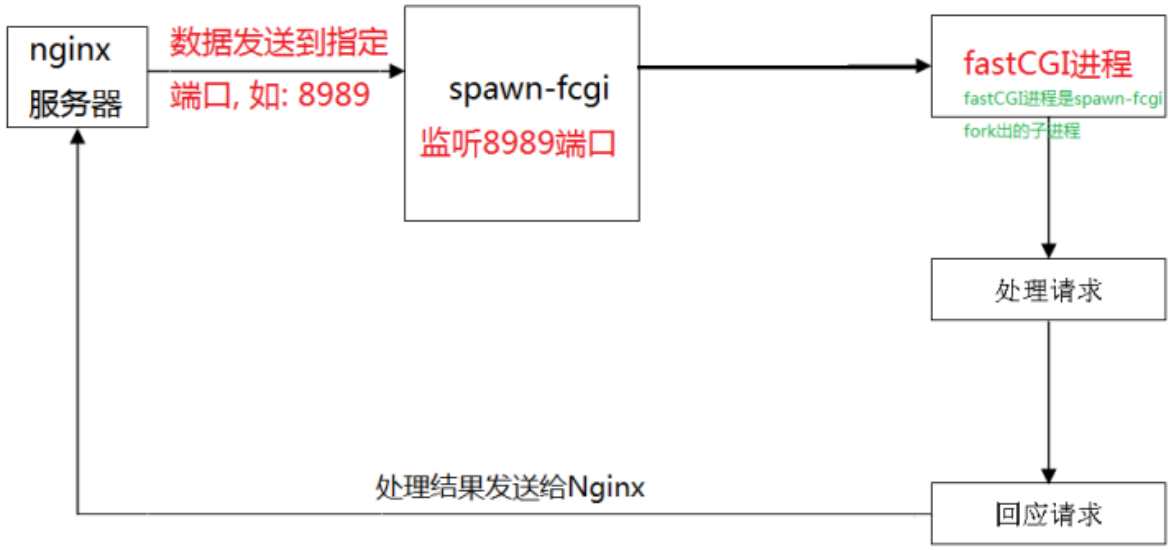
```
kiko@Hkiko:~/lesson/fcgi-2.4.1-SNAP-0910052249/examples$ ls
authorizer      authorizer.o  echo-cpp      echo-cpp.o  echo-x     echo-x.o    log-dump.o   Makefile.in  size.mak  threaded.c
authorizer.c    echo          echo-cpp.cpp  echo.mak    echo-x.c   log-dump    Makefile     size         size.o    threaded-threaded.o
authorizer.mak  echo.c        echo-cpp.mak  echo.o      echox.mak  log-dump.c  Makefile.am  size.c       threaded
kiko@Hkiko:~/lesson/fcgi-2.4.1-SNAP-0910052249/examples$ gcc -o app echo.c
/usr/bin/ld: /tmp/cc0W4nxv.o: in function `PrintEnv':
echo.c:(.text+0x28): undefined reference to `FCGI_printf'
/usr/bin/ld: echo.c:(.text+0x45): undefined reference to `FCGI_printf'
/usr/bin/ld: echo.c:(.text+0x67): undefined reference to `FCGI_printf'
/usr/bin/ld: /tmp/cc0W4nxv.o: in function `main':
echo.c:(.text+0xbe): undefined reference to `FCGI_printf'
/usr/bin/ld: echo.c:(.text+0xfe): undefined reference to `FCGI_printf'
/usr/bin/ld: /tmp/cc0W4nxv.o:echo.c:(.text+0x111): more undefined references to `FCGI_printf' follow
/usr/bin/ld: /tmp/cc0W4nxv.o: in function `main':
echo.c:(.text+0x11f): undefined reference to `FCGI_getchar'
/usr/bin/ld: echo.c:(.text+0x139): undefined reference to `FCGI_printf'
/usr/bin/ld: echo.c:(.text+0x145): undefined reference to `FCGI_putchar'
/usr/bin/ld: echo.c:(.text+0x162): undefined reference to `FCGI_printf'
/usr/bin/ld: echo.c:(.text+0x190): undefined reference to `FCGI_Accept'
collect2: error: ld returned 1 exit status
kiko@Hkiko:~/lesson/fcgi-2.4.1-SNAP-0910052249/examples$ gcc -o app echo.c -lfcgi
kiko@Hkiko:~/lesson/fcgi-2.4.1-SNAP-0910052249/examples$ ldd app
        linux-vdso.so.1 (0x00007ffd315e2000)
        libfcgi.so.0 => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007efdb32af000)
        /lib64/ld-linux-x86-64.so.2 (0x00007efdb34b9000)
```

在fastCGI的安装包有examples目录，其中有官方给出的多个例子，这里使用echo.c来生成可执行程序，其中要用-l指定库名称。此外，要将.so动态库放到可以搜索到的路径。否则在使用spawn命令启动app可执行程序时会出现问题

4.4 nginx和fastCGI协同开发（动态请求）

- 说明

> nginx 不能像apache那样直接执行外部可执行程序，但nginx可以作为代理服务器，将请求转发给后端服务器， 这也是nginx的主要作用之一。其中nginx就支持FastCGI代理，接收客户端的请求，然后将请求转发给后端fastcgi 进程。下面介绍如何使用C/C++编写cgi/fastcgi，并部署到nginx中。 通过前面的介绍知道，fastcgi进程由FastCGI进程管理器管理，而不是nginx。这样就需要一个FastCGI管理，管理我们编写fastcgi程序。我们使用spawn-fcgi作为FastCGI进程管理器。 spawn-fcgi是一个通用的FastCGI进程管理器，简单小巧，原先是属于lighttpd的一部分，后来由于使用比较广 泛，所以就迁移出来作为独立项目了。spawn-fcgi使用pre-fork 模型， 功能主要是打开监听端口，绑定地址，然 后fork-and-exec创建我们编写的fastcgi应用程序进程，退出完成工作 。fastcgi应用程序初始化，然后进入死循环 侦听socket的连接请求。



```
请求 = http://localhost/login?name=zhangsan&age=12&gender=man
1. 客户端访问, 发送请求
2. nginx web服务器, 无法处理用户提交的数据
3. spawn-fcgi - 通信过程中的服务器角色
        被动接收数据
        在spawn-fcgi启动的时候给其绑定IP和端口
4. fastCGI程序
        程序员写的 -> login.c -> 可执行程序( login )
        使用 spawn-fcgi 进程管理器启动 login 程序, 得到一进程
```

- 1、nginx的数据转发 - 修改nginx的配置文件nginx.conf

```
通过请求的url http://localhost/login?user=zhangsan&password=123456
指令:
    - 去掉协议
    - 去掉域名/IP + 端口
    - 如果尾部有文件名 去掉
    - 去掉 ? 和后面的字符串（请求参数）
    - 剩下就是指令 /login
login /login
{
    # 转发这个数据，fastCGI进程
    fastcgi_pass 地址信息:端口;

    # fastcgi.conf和nginx.conf在同一级目录： /usr/local/nginx/conf
    # 这个文件中定义了http通信的时候用到的环境变量，nginx赋值的
    include fastcgi.conf;
}

扩展：关于动态请求location配置指令，而不是服务器资源的本地目录路径。
在nginx中如果客户端请求的是动态请求，那么对于location的请求路径中配置的路由不用本地有任何对应的目录或者文件。
/login      --> 可以看作一个指令，而不是目录或者文件。
/user/login  --> user和login也都是指令，不对应任何文件或者目录。
此外，不必担心在后面添加一个/，也就是/login/、/user/login/也是一样的。
```

fastcgi.conf

```
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;
fastcgi_param  QUERY_STRING       $query_string;
fastcgi_param  REQUEST_METHOD     $request_method;
fastcgi_param  CONTENT_TYPE       $content_type;
fastcgi_param  CONTENT_LENGTH     $content_length;

fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;
fastcgi_param  REQUEST_URI        $request_uri;
fastcgi_param  DOCUMENT_URI       $document_uri;
fastcgi_param  DOCUMENT_ROOT      $document_root;
fastcgi_param  SERVER_PROTOCOL    $server_protocol;
fastcgi_param  REQUEST_SCHEME     $scheme;
fastcgi_param  HTTPS              $https if_not_empty;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;

# PHP only, required if PHP was built with --enable-force-cgi-redirect
fastcgi_param  REDIRECT_STATUS    200;

```

- 2、spawn-fcgi启动

```
# 前提条件: 程序员的fastCGI程序已经编写完毕 -> 可执行文件 login
spawn-fcgi -a IP地址 -p 端口 -f fastcgi可执行程序
- IP地址: 应该和nginx的 fastcgi_pass 配置项对应
    - nginx: localhost -> IP: 127.0.0.1
    - nginx: 127.0.0.1 -> IP: 127.0.0.1
    - nginx: 192.168.1.100 -> IP: 192.168.1.100
- 端口:
    应该和nginx中nginx.conf配置文件的的 fastcgi_pass 中的端口一致
```

- 3、fastCGI程序编写，要在spawn-fcgi启动之前编写

```
// http://localhost/login?user=zhang3&passwd=123456&age=12&sex=man
// 要包含的头文件
#include "fcgi_config.h" // 可选
#include "fcgi_stdio.h" // 必须的, 编译的时候找不到这个头文件, find->path , gcc -I

/*
    genenv(char *): 获取http通信的时候用到的环境变量, 在fastcgi.conf中定义的
                通过获取的环境变量可以获取来自nginx转发的请求的相关请求参数等等。
*/

// 编写代码的流程
int main()
{
    // FCGI_Accept()是一个阻塞函数, nginx给fastcgi程序发送数据的时候解除阻塞
```

```
    while (FCGI_Accept() >= 0)
    {
        // 1. 接收数据
        // 1.1 get方式提交数据 - 数据在请求行的第二部分
        // user=zhang3&passwd=123456&age=12&sex=man
        char *text = getenv("QUERY_STRING");

        // 1.2 post方式提交数据
        char *contentLength = getenv("CONTENT_LENGTH");

        // 根据长度大小判断是否需要循环
        // 2. 按照业务流程进行处理
        // ...

        // 3. 将处理结果发送给nginx
        // 数据回发的时候, 需要告诉nginx处理结果的格式 - 假设是html格式
        printf("Content-type: text/html\r\n");
        printf("<html>处理结果</html>");
    }
}
```

- GET请求的http通信的环境变量

```
假设请求：http://localhost/login?name=zhangsan&password=123456

# 设置FastCGI应用程序所需的脚本文件路径。 /usr/local/nginx/login
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;

# 设置客户端请求的URI中的查询字符串部分(GET请求才有)。name=zhangsan&password=123456
fastcgi_param  QUERY_STRING       $query_string;

# 请求方法GET/POST/...
fastcgi_param  REQUEST_METHOD     $request_method;

# 设置客户端请求的Conten-Type。通常在POST请求方法中设置。
# 1、application/x-www-form-urlencoded
# 2、application/json
# 3、text/xml
# 4、multipart/form-data
fastcgi_param  CONTENT_TYPE       $content_type;

# 设置客户端请求的Content-Length。通常在POST请求中使用，在GET请求中没有设置
fastcgi_param  CONTENT_LENGTH     $content_length;

# 脚本名 /login
fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;

# 请求的uri。GET请求：/login?name=zhangsan&password=123456。POST请求：/login
fastcgi_param  REQUEST_URI        $request_uri;
```

```
# 客户端和服务器的地址和端口
fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;

# PHP only, required if PHP was built with --enable-force-cgi-redirect
fastcgi_param  REDIRECT_STATUS    200;
```

GET请求

```
FCGI_ROLE=RESPONDER
SCRIPT_FILENAME=/usr/local/nginx/html/mytest
QUERY_STRING=username=tom&phone=123&email=hello%40qq.com&date=2018-01-01&sex=female&class=3&rule=on
REQUEST_METHOD=GET
CONTENT_TYPE=
CONTENT_LENGTH=
SCRIPT_NAME=/mytest
REQUEST_URI=/mytest?username=tom&phone=123&email=hello%40qq.com&date=2018-01-01&sex=female&class=3&rule=on
DOCUMENT_URI=/mytest
DOCUMENT_ROOT=/usr/local/nginx/html
SERVER_PROTOCOL=HTTP/1.1
REQUEST_SCHEME=http
GATEWAY_INTERFACE=CGI/1.1
SERVER_SOFTWARE=nginx/1.10.1
REMOTE_ADDR=192.168.247.1
REMOTE_PORT=51865
SERVER_ADDR=192.168.247.135
SERVER_PORT=80
SERVER_NAME=localhost
REDIRECT_STATUS=200
HTTP_HOST=192.168.247.135
HTTP_CONNECTION=keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS=1
HTTP_USER_AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT_LANGUAGE=zh,zh-CN;q=0.9,en;q=0.8
```

POST请求

```
FCGI_ROLE=RESPONDER
SCRIPT_FILENAME=/usr/local/nginx/html/mytest
QUERY_STRING=
REQUEST_METHOD=POST
CONTENT_TYPE=application/x-www-form-urlencoded
CONTENT_LENGTH=86
SCRIPT_NAME=/mytest
REQUEST_URI=/mytest
DOCUMENT_URI=/mytest
DOCUMENT_ROOT=/usr/local/nginx/html
SERVER_PROTOCOL=HTTP/1.1
REQUEST_SCHEME=http
GATEWAY_INTERFACE=CGI/1.1
SERVER_SOFTWARE=nginx/1.10.1
REMOTE_ADDR=192.168.247.1
REMOTE_PORT=52293
SERVER_ADDR=192.168.247.135
SERVER_PORT=80
SERVER_NAME=localhost
REDIRECT_STATUS=200
HTTP_HOST=192.168.247.135
HTTP_CONNECTION=keep-alive
HTTP_CONTENT_LENGTH=86
HTTP_CACHE_CONTROL=max-age=0
HTTP_UPGRADE_INSECURE_REQUESTS=1
HTTP_ORIGIN=null
HTTP_CONTENT_TYPE=application/x-www-form-urlencoded
HTTP_USER_AGENT=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT_LANGUAGE=zh,zh-CN;q=0.9,en;q=0.8
```

4.5 Post提交数据的常用方式

> Http协议规定 POST 提交的数据必须放在消息主体（entity-body）中，但协议并没有规定数据必须 使用什么编码方式。
>
> 开发者完全可以自己决定消息主体的格式
>
> 数据发送出去，还要服务端解析成功才有意义, 服务端通常是根据请求头（headers）中的 Content-Type 字段来获知请求中的消息主体是用何种方式编码，再对主体进行解析。

```
# 1、application/x-www-form-urlencoded

# 请求行
POST http://www.example.com HTTP/1.1
# 请求头
Content-Type: application/x-www-form-urlencoded;charset=utf-8
# 空行
# 请求数据(向服务器提交的数据)
title=test&user=kevin&passwd=32222
```

```
# 2、application/json

POST / HTTP/1.1
Content-Type: application/json;charset=utf-8
{"title":"test","sub":[1,2,3]}
```

```
# 3、text/xml

POST / HTTP/1.1
Content-Type: text/xml
<?xml version="1.0" encoding="utf8"?>
<methodcall>
    <methodname color="red">examples.getStateName</methodname>
    <params>
        <value><i4>41</i4></value>
    </params>
</methodcall>
<font color="red">nihao, shijie</font>
```

```
4、multipart/form-data

POST / HTTP/1.1
Content-Type: multipart/form-data
# 发送的数据
------WebKitFormBoundaryPpL3BfPQ4cHShsBz \r\n
Content-Disposition: form-data; name="file"; filename="qw.png"
Content-Type: image/png\r\n; md5="xxxxxxxxxx"
\r\n
.............文件内容.................
.............文件内容.................
```

```
------WebKitFormBoundaryPpL3BfPQ4cHShsBz--
Content-Disposition: form-data; name="file"; filename="qw.png"
Content-Type: image/png\r\n; md5="xxxxxxxxxx"
\r\n
.............文件内容.................
.............文件内容.................
------WebKitFormBoundaryPpL3BfPQ4cHShsBz--
```