# 前言：

由于本人工作原因，涉及到网络直播领域，其中视频的回放下载，涉及到了一些视频下载方面的技术。针对于一个完整视频的下载，目前市面上的主流做法是，先将整个视频流切片，存储到文件服务器中，在用户需要观看回放视频时。通过一个视频回源服务器，去文件服务器中逐个请求切片，返回给用户播放。

今天着重探讨的是关于回源服务器缓存的配置以及合理的缓存策略。

通过给回源服务器配置缓存的案例，详细讲解一整套缓存配置机制，并且可沿用到其他任何缓存配置场景中。

今天的讲解分为四点：

**回源服务器的工作是啥**

**为啥需要给回源服务器加缓存**

**如何配置缓存**

**如何针对业务场景配置完备的缓存机制**

回源服务器的工作：

回源服务器在下面叙述中简称：源站

如图所示，在文件下载的过程中，横跨在cdn与文件服务器之间，作为下载枢纽。

源站架构：源站是nginx+php的webserver架构，如图所示：

但如果源站只是简单的收到请求，然后下载资源，再返回，势必会存在以下几点不够优化的问题：

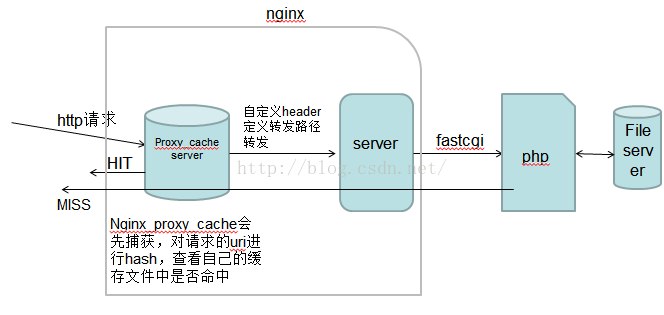
1、cdn可能存在多次回源现象

2、源站对同一资源的多次下载，存在网络流量带宽浪费，以及不必要的耗时。

所以为了优化这些问题，需要给源站做一层缓存。缓存策略采用nginx自带的proxy\_cache模块。

proxy\_cache原理：

proxy\_cache模块的工作原理如图所示：



如何配置proxy\_cache模块

在nginx.conf文件中添加如下代码：

http{

......

proxy\_cache\_path/data/nginx/tmp-test levels=1:2 keys\_zone=tmp-test:100m inactive=7d max\_size=1000g;

}

代码说明：

proxy\_cache\_path 缓存文件路径

levels 设置缓存文件目录层次；levels=1:2 表示两级目录

keys\_zone 设置缓存名字和共享内存大小

inactive 在指定时间内没人访问则被删除

max\_size 最大缓存空间，如果缓存空间满，默认覆盖掉缓存时间最长的资源。

当配置好之后，重启nginx，如果不报错，则配置的proxy\_cache会生效

查看  proxy\_cache\_path /data/nginx/目录，

会发现生成了tmp-test文件夹。

如何使用proxy\_cache

在你对应的nginx vhost server配置文件中添加如下代码:

location /tmp-test/ {

proxy\_cache tmp-test;

proxy\_cache\_valid 200 206 304 301 302 10d;

proxy\_cache\_key $uri;

proxy\_set\_header Host $host:$server\_port;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_passhttp://127.0.0.1:8081/media\_store.php/tmp-test/;

}

配置项介绍：

Proxy\_cache tmp-test 使用名为tmp-test的对应缓存配置

proxy\_cache\_valid  200 206 304 301 302 10d; 对httpcode为200…的缓存10天

proxy\_cache\_key $uri  定义缓存唯一key,通过唯一key来进行hash存取

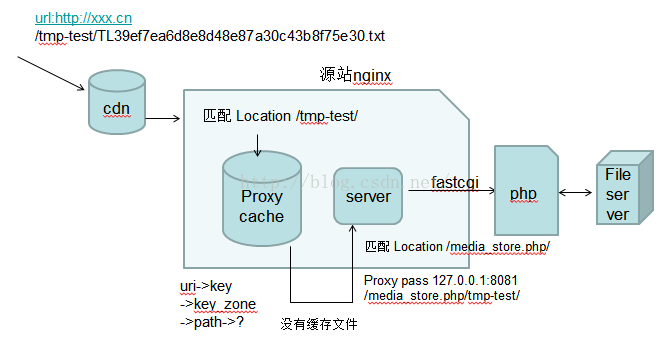
proxy\_set\_header  自定义http header头，用于发送给后端真实服务器。

proxy\_pass  指代理后转发的路径，注意是否需要最后的/

到这里，最基本的proxy\_cache功能就配置成功了。当uri成功匹配到该location,则proxy\_cache就会生效。

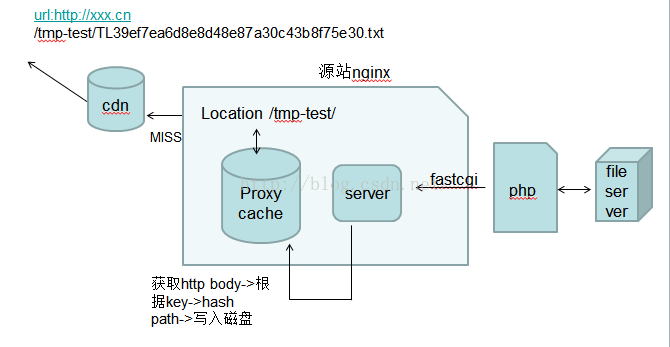
## 添加proxy\_cache之后，请求过程的变化:

1、第一次访问：

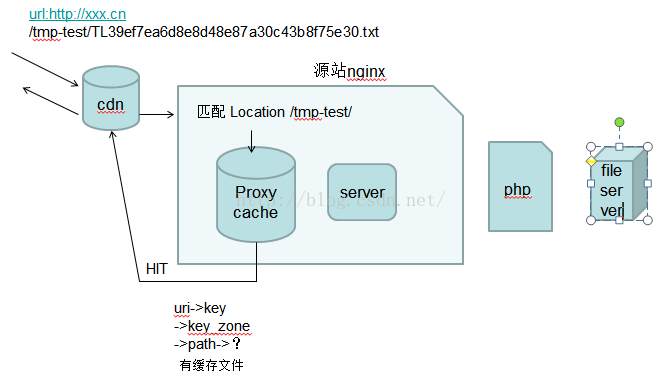


第一次访问，proxy\_cache并没有找到对应的缓存文件（未命中缓存MISS），所以当第一次请求完成的同时，proxy\_cache会保持缓存：

2、保存缓存，如图所示：



3、同一个url第二次访问，当同一个文件再次到达源站，proxy\_cache就会找到其对应的缓存文件（命中缓存HIT）直接返回给请求端，无需再执行php程序，如图所示：



提出疑问：

到此，就完成了最基本的proxy\_cache配置和访问过程介绍，但是最基本的配置，往往无法满足我们的业务需求，我们往往会提出以下几点疑问和需求：

需要主动清理缓存文件

写入路径为一块磁盘，如果磁盘打满该怎么解决？

如何让源站支持断点续传，以及断点续传的缓存策略

如果请求端 range 请求（分片下载）一个大资源，同样的uri，如何区别请求？

还需要告诉请求端，资源的过期时间

日志统计，如何配置命中与不命中字段，如何做统计？

面对以上疑问，我们一个一个解决。

## 问题一：主动清理缓存

采用：nginx  proxy\_cache\_purge 模块 ，该模块与proxy\_cache成对出现，功能正好相反。

设计方法：在nginx中，另启一个server，当需要清理响应资源的缓存时，在本机访问这个server。

例如:

访问 127.0.0.1:8083/tmp-test/TL39ef7ea6d8e8d48e87a30c43b8f75e30.txt 即可清理该资源的缓存文件。

配置方法：

location /tmp-test/ {

allow 127.0.0.1; //只允许本机访问

deny all; //禁止其他所有ip

proxy\_cache\_purge tmp-test $uri; //清理缓存

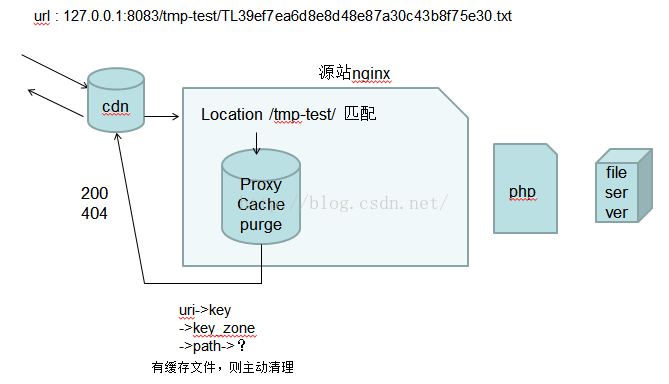
}

proxy\_cache\_purge：缓存清理模块

tmp-test：指定的key\_zone

$uri：指定的生成key的参数

proxy\_cache\_purge缓存清理过程，如图所示：



## 问题二：缓存文件强磁盘打满该怎么办？

由于写入路径为一个单一目录，只能写入一块磁盘。一块磁盘很快就会被打满，解决该问题有如下两种方法：

1、将多块磁盘做磁盘阵列？ 缺点是：减小了实际的存储空间。

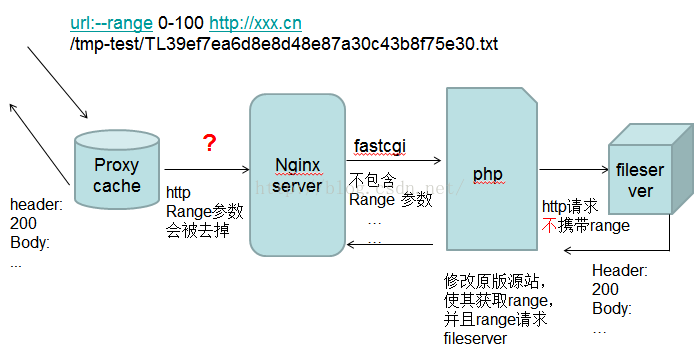
2、巧妙得运用proxy\_cache\_path的目录结构，由于levels=1:2，这导致缓存文件的目录结构为两层，每层目录名，都是由hash函数生成。如图所示：

总共含有16\*16\*16=4096个文件目录。对该一级目录进行软连接，分别将0-f软连接到你所需要的指定磁盘目录上，如图所示：

通过软链的方法，实现：将不同盘下的目录作为真正存放数据的路径，解决了多盘利用，单盘被打满的问题。

## 问题三：支持range（断点续传）

添加上缓存代理之后，客户端发起的range请求将会失效，如下图所示：



导致range参数无法传递到下一级的原因如下：

当缓存代理转发http请求到后端服务器时，http header会改变，header中的部分参数，会被取消掉。其中range参数被取消，导致，后端nginx服务器没有收到range参数，最终导致这个分片下载不成功。所以需要对代理转发的header进行配置。

例如：

location /tmp-test/ {

proxy\_cache tmp-test;

proxy\_cache\_valid 200 206 304 301 302 10d;

proxy\_cache\_key $uri;

<span style="color:#ff0000;">proxy\_set\_header Range $http\_range;</span>

proxy\_pass http://127.0.0.1:8081/media\_store.php/tmp-test/;

}

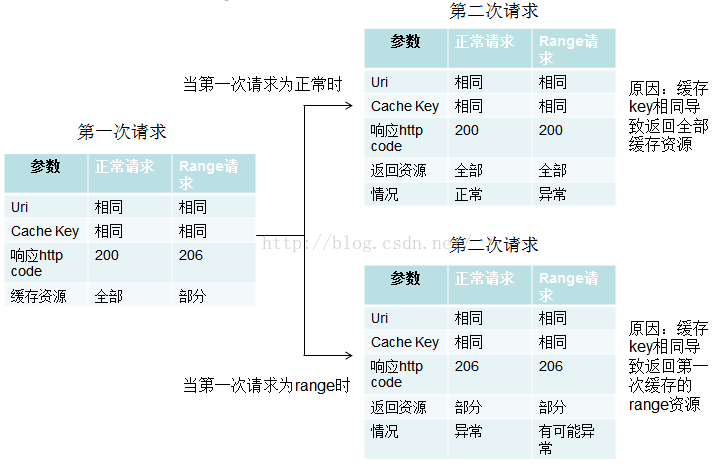
红色部分的含义：将http请求中的range值（$http\_range）放到代理转发的http请求头中作为参数range的值。

## 问题四，当支持range加载后，proxy\_cache\_key，则需要重新配置：

如果请求端 Range请求（分片下载）一个大资源，同样的uri，proxy cache如何识别资源对应的key。

由于nginx配置为：proxy\_cache\_key $uri，用uri作为key

所以当请求为普通请求和range请求时，都是同样的uri作为key。proxy\_cache将有可能导致错误返回。如下图所示：



解决方法如下：

修改proxy\_cache\_key ，配置proxy\_cache\_key $http\_range$uri;

这样就能解决：key唯一性。可以避免不管是正常请求还是不同的range请求，第一次获取的内容和之后获取的缓存内容都不会出现异常。

## 问题五：如何配置-返回过期时间

需要通过返回过期时间来指定请求端，哪些资源需要缓存，哪些资源不缓存，

参数 正常请求 range请求

返回过期时间 返回 不返回

为了防止请求端将分片资源当做完整资源缓存起来，我们需要对正常请求，返回过期时间；对range请求， 不返回过期时间。

解决该问题，通过对nginx配置即可解决：

location /media\_store.php {

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index media\_store.php;

fastcgi\_param SCRIPT\_FILENAME $document\_root/$fastcgi\_script\_name;

include fastcgi\_params;

if ( $http\_range = ''){

expires 2592000s;

}

}

在proxy\_pass代理之后的location中加入对$http\_range的判断，expires 表示过期时间。 2592000s指缓存过期时间。

## 问题七：缓存命中情况如何在http头中体现，以及在nginx日志中查看

解决方法：

利用nginx $upstream\_cache\_status变量:该变量代表缓存命中的状态，

如果命中，为HIT；如果未命中，为MISS

在返回nginx server配置中添加：

add\_header  Nginx-Cache "$upstream\_cache\_status";

在nginxlog中添加：

log\_format       combinedio  …$upstream\_cache\_status;

http返回head截图：

nginx log日志截图：

总结：

整个一套完备的缓存策略就介绍到此，这套方案中不仅实现了基本的缓存配置，还解决了实际场景应用中会遇到的，磁盘扩展，缓存清理，断点续传，缓存过期时间，缓存命中提示等问题，只要将这套方案灵活运用，不管是再复杂的场景，基本都能满足需求。以上都是我在工作中爬过的坑，不断完善总结出的结果，希望对读者能有帮助。