一个缓存配置案例

2021 年 3 月 5 日

1 问题

我们有一个专门输出 (serving) 静态文件 (static files) 的后端记做 B，一个网页服务器记做 W，对于所有来自前端 C 的请求 $\{Q\}$，我们希望在 W 中缓存 (caching) Q，具体地：对每一个这样的 Q，W 首先检查自己命名空间中的缓存对象，如果缓存对象表示 Q 匹配，则 W 向缓存对象中取出 F，并且代替 B，直接将 F 作为 Q 的响应 R，返还给 C；如果 W 发现 Q 不匹配，则 W 向 B 请求 F，B 将 F 返还给 W，然后 W 用 F 更新缓存，与此同时将 F 发回给 C．简单来说，这样做，是拿 W 保护 B，事实上，B 只接受来自 W 的请求，这样一来，所有的请求都会经过 W，当面临大规模的请求时，由于有 W 的缓存起作用，所以真正发往 B 的请求 $\{Q\}$ 的数量会很少，这正是我们希望实现的．

2 初始解

对于 W，我们使用的是 NginX，版本是 `1.18.0`，首先，我们在 `/etc/nginx/nginx.conf` 的 `http` 块中，加入如下语句：

```
proxy_cache_path
/var/www/cache
levels=1:2
keys_zone=my_cache:10m
max_size=1g
use_temp_path=off;
```

这时我们对于全局的 `http` 请求声明了一个缓存路径,具体则规定了缓存应该存在哪里 (`/var/www/cache`)，缓存目录的目录层级结构 (`levels=1:2`)，缓存元数据 (meta-data) 例如缓存键和使用情况在内存中的内存大小限制 (`my_cache:10m`)，缓存内容在磁盘中的使用空间限制`1g`．

这里的 `\levels` 最多可以有三个层级，这里我们用了两个．NginX 会将缓存键首先进行 MD5 运算，得到一个散列值的 16 进制字符串表示，例如：

```
047373d73f4f141840dee438fe343268
```

就是一个缓存键经过散列处理后得到的序列．我们设置了 `levels=1:2`, 于是，NginX 首先提取字符串的最后一个字符 (对应 `levels=1:2` 中的 `1`)，在这里是 `8`，作为第一层文件夹的文件名，然后，再往左读入两个字符 `26` 作为第二层文件夹的文件名，读入两个字符是对应 `levels=1:2` 中的 `2`，然后再创建一个名为 `047373d73f4f141840dee438fe343268` 的文件放在 `8/26` 目录中（如果不存在的话），如图 2-1所示：

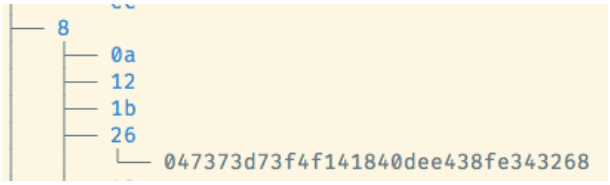

图 2-1: 目录结构图示

显然，当缓存键的个数特别多的时候，如果有多个层级，查询起来会比较快，当缓存键的个数比较少的时候，多个层级的加速效果不明显．

创建好缓存空间 (在磁盘上的目录会由 NginX 自动创建) 之后，我们在一个`server` 内，添加如下配置块：

```
location ~* \.(jpe?g|png|gif|ico|svg|pdf|md|mp4|css)$ {
    proxy_cache my_cache;
    rewrite ^(.*)$ /static$1 break;
    proxy_pass https://domain/path;
}
```

并且执行

```
nginx -s reload
```

以更新配置．然后我们进入`/var/www/cache` 目录查看，发现该目录空空如也．为了进一步确认，我们在要缓存的`location` 中，添加如下语句：

```
add_header X-Cache-Status-Nginx $upstream_cache_status;
```

并且执行命令更新配置．在浏览器的开发者调试工具中，发现了：

```
x-cache-status-nginx: MISS
```

并且连续请求该文件多次都是如此，这说明缓存并没有生效．

3 调查

通过查阅资料，我们在 NginX 的官方网站的一篇文章Nginx Caching Guide发现了图 3-1：

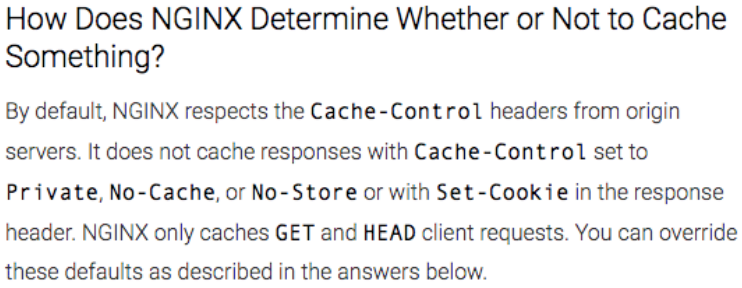

图 3-1: 说明

可能是来自后端 B 的 HTTP 响应头 (headers) 中的某些个字段使得 NginX 决定不进行缓存．但是当我们绕过中间服务器 W 直接请求后端 B 的时候，我们发现 HTTP 响应的 headers 中并没有出现 Cache-Control，也没有 Set-Cookie 或者其他任何与缓存控制相关的头部．

接下来，我们阅读了RFC7234，在第 6 页找到了答案：首先文中指出一个 Proxy 是默认不进行缓存的，仅当多个条件同时满足时，才进行缓存，这其中就包括了来自被代理的服务器的响应或者包含一个 Expires 字段，或者包含一个 max-age 字段，或者包含一个 s-max-age 字段，或者包含 Cache-Control 字段（并且其中的值允许响应内容被缓存），如果这些都不成立，则 Proxy 可以 (MAY) 考虑根据 Last-Modified 字段使用启发式方法计算缓存有效期.

也就是说，源服务器也就是后端 B 给出的响应的头部中并没有指示应该如何进行缓存，那么从而导致了，中间的 Proxy 服务器不知道如何计算缓存有效期，那么解决的办法就是，要么我们设置中间的 W，使得它给所有来自后端 B 的内容显示地设置一个缓存有效期，要么从源头解决问题，去配置后端的 B 使得它自己往它输出的内容添加有效的缓存控制字段.

4 解决方案

我们决定在中间的 Proxy 服务器 B 进行配置，具体是在要被缓存的`location` 块中加入：

```
proxy_cache_valid any 1w;
```

完整的配置如下：

```
location ~* \.(jpe?g|png|gif|ico|svg|pdf|md|mp4)$ {
    proxy_cache my_cache;
    rewrite ^(.*)$ /static$1 break;
    proxy_pass https://beyondstars-blog-statics.s3.amazonaws.com;

    proxy_cache_valid any 1w;

    add_header X-Cache-Status-Nginx $upstream_cache_status;
}
```

这样就解决了问题.

另外一种解决方案是后端 B 在自己输出的内容的头部加入：

```
Cache-Control: public; max-age=259200
```

这样中间的 W 就知道该如何缓存了.