

web 服务器-Nginx

一. 讲在 Nginx 之前

同步与异步:

同步与异步的重点在消息通知的方式上, 也就是调用结果的通知方式不同。

同步: 当一个同步调用发出去后, 调用者要一直等待调用的结果通知后, 才能进行后续的执行。

异步: 当一个异步调用发出去后, 调用者不必一直等待调用结果的返回, 异步调用, 要想获得结果, 一般有两种方式:

- 1、主动轮询异步调用的结果;
- 2、被调用方通过 callback (回调通知) 来通知调用方调用结果。

实例解释:

同步取快递: 小明收到快递将送达的短信, 在楼下一直等到快递送达。

异步取快递: 小明收到快递将送达的短信, 快递到楼下后, 小明再下楼去取。

异步取快递, 小明知道快递到达楼下有两种方式:

- 1、不停的电话问快递小哥到了没有, 即主动轮询;
- 2、快递小哥到楼下后, 打电话通知小明, 然后小明下楼取快递, 即回调通知。

阻塞与非阻塞:

阻塞与非阻塞的重点在于进/线程等待消息时候的行为, 也就是在等待消息的时候, 当前进/线程是挂起状态, 还是非挂起状态。

阻塞: 调用在发出去后, 在消息返回之前, 当前进/线程会被挂起, 直到有消息返回, 当前进/线程才会被激活

非阻塞: 调用在发出去后, 不会阻塞当前进/线程, 而会立即返回。

实例解释:

阻塞取快递: 小明收到快递即将送达的信息后, 什么事都不做, 一直专门等快递。

非阻塞取快递: 小明收到快递即将送达的信息后, 等快递的时候, 还一边敲代码、一边刷微信。

同步与异步, 重点在于消息通知的方式; 阻塞与非阻塞, 重点在于等消息时候的行为。

所以, 就有了下面 4 种组合方式

- **同步阻塞:** 小明收到信息后, 啥都不干, 等快递;
- **同步非阻塞:** 小明收到信息后, 边刷微博, 边等着取快递;
- **异步阻塞:** 小明收到信息后, 啥都不干, 一直等着快递员通知他取快递;
- **异步非阻塞:** 小明收到信息后, 边刷着微博, 边等快递员通知他取快递。

大部分程序的 I/O 模型都是同步阻塞的, 单个进程每次只在一个文件描述符上执行 I/O 操作, 每次 I/O 系统调用都会阻塞, 直到完成数据传输。传统的服务器采用的就是同步阻塞的多进程模型。一个 server 采用一个进程负责一个 request 的方式, 一个进程负责一个 request, 直到会话结束。进程数就是并发数, 而操作系统支持的进程数是有限的, 且进程数越多, 调度的开销也越大, 因此无法面对高并发。

Nginx 采用了异步非阻塞的方式工作。我们先来了解一下 I/O 多路复用中的 epoll 模型。

epoll 模型:

当连接有 I/O 事件产生的时候, epoll 就会去告诉进程哪个连接有 I/O 事件产生, 然后进程就去处理这个事件。

例如: 小明家楼下有一个收发室, 每次有快递到了, 门卫就先代收并做了标记; 然后通知小明去取送给小明的快递。

为什么 Nginx 比其他 web 服务器并发高 (Nginx 工作原理):

Nginx 配置 use epoll 后, 以异步非阻塞方式工作, 能够轻松处理百万级的并发连接。

处理过程: 每进来一个 request, 会有一个 worker 进程去处理。但不是全程的处理, 处理到可能发生阻塞的地方。比如向后端服务器转发 request, 并等待请求返回。那么, 这个处理的 worker 不会这么傻等着, 他会在发送完请求后, 注册一个事件: “如果后端服务器返回了, 告诉我一声, 我再接着干”。于是他就休息去了。此时, 如果再有新的 request 进来, 他就可以很快再按这种方式处理。而一旦后端服务器返回了, 就会触发这个事件, worker 才会来接手, 这个 request 才会接着往下走。通过这种快速处理, 快速释放请求的方式, 达到同样的配置可以处理更大并发量的目的。

二. Nginx 详解

1. 概述

Nginx (engine x) 是一个高性能的 HTTP 和反向代理 web 服务器, 同时也提供了 IMAP/POP3/SMTP 服务。Nginx 是由伊戈尔·赛索耶夫为俄罗斯访问量第二的 Rambler.ru 站点开发的, 第一个公开版本 0.1.0 发布于 2004 年 10 月 4 日。

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件 (IMAP/POP3) 代理服务器, 在 BSD-like 协议下发行。其特点是占有内存少, 并发能力强。

2. 工作模式

nginx 有两种工作模式: master-worker 模式和单进程模式。在 master-worker 模式下, 有一个 master 进程和至少一个的 worker 进程, 单进程模式顾名思义只有一个进程。这两种模式有各自的特点和适用场景。

master-worker:

该模式下, nginx 启动成功后, 会有一个 master 进程和至少一个的 worker 进程。master 进程负责处理

系统信号，加载配置，管理 worker 进程（启动，杀死，监控，发送消息/信号等）。worker 进程负责处理具体的业务逻辑，也就是说，对外部来说，真正提供服务的是 worker 进程。生产环境下一般使用这种模式，因为这种模式有以下优点：

1. 稳定性高，只要还有 worker 进程存活，就能够提供服务，并且一个 worker 进程挂掉 master 进程会立即启动一个新的 worker 进程，保证 worker 进程数量不变，降低服务中断的概率。
2. 配合 linux 的 cpu 亲和性配置，可以充分利用多核 cpu 的优势，提升性能
3. 处理信号/配置重新加载/升级时可以做到尽可能少或者不中断服务（热重启）

单进程模式：

单进程模式下，nginx 启动后只有一个进程，nginx 的所有工作都由这个进程负责。由于只有一个进程，因此可以很方便地利用 gdb 等工具进行调试。该模式不支持 nginx 的平滑升级功能，任何的信号处理都可能造成服务中断，并且由于是单进程，进程挂掉后，在没有外部监控的情况下，无法重启服务。因此，该模式一般只在开发阶段和调试时使用，生产环境下不会使用。（了解）

3. 配置文件结构

```
user www www;
#程序运行用户和组

worker_processes auto;
#启动进程，指定 nginx 启动的工作进程数量，建议按照 cpu 数目来指定，一般等于 cpu 核心数目

error_log /home/wwwlogs/nginx_error.log crit;
#全局错误日志

pid /usr/local/nginx/logs/nginx.pid;
#主进程 PID 保存文件

worker_rlimit_nofile 51200;
#文件描述符数量

events
{
    use epoll;
    #使用 epoll 模型，对于 2.6 以上的内核，建议使用 epoll 模型以提高性能
    worker_connections 51200;
    #工作进程的最大连接数量
}

http{
    #网站优化参数

    server {
        #具体的某一网站的配置信息
        listen 80;
        #监听端口
        root html;
        #网页根目录 (/usr/local/nginx/html)
        server_name www.atguigu.com;
        #服务器域名
        index index.html;
        #默认加载页面
        access_log logs/access.log;
        #访问日志保存位置
        .....;
    }
}
```

```
location (.*).php$ {  
    用正则匹配具体的访问对象;  
}  
location {  
    跳转等规则;  
}  
}  
server {  
    虚拟主机;  
}  
}
```

4. Nginx 相关实验

<注意事项>

1. 注意配置文件中的结尾有;作为结束~！（切记！）
2. 每次实验修改完配置文件后需要重启 nginx 才会生效

```
# pkill -HUP nginx
```

实验 1: Nginx 的状态统计

- a、安装 nginx 时将 --with-http_stub_status_module 模块开启
- b、修改 nginx 配置文件（写入要访问的 server 标签中）

```
location /nginx_status{  
    stub_status on;  
    access_log off;  
}
```

- c、客户端访问网址:http://IP/nginx_status

"Active connections"表示当前的活动连接数;

"server accepts handled requests"表示已经处理的连接信息

三个数字依次表示已处理的连接数、成功的 TCP 握手次数、已处理的请求数

实验 2: 目录保护

- a、原理和 apache 的目录保护原理一样（利用上一个实验接着完成）
- b、在状态统计的 location 中添加:

```
auth_basic "Welcome to nginx_status!";  
auth_basic_user_file /usr/local/nginx/html/htpasswd.nginx;
```

- c、使用 http 的命令 htpasswd 进行用户密码文件的创建（生成在上面指定的位置）

```
# htpasswd -c /usr/local/nginx/html/htpasswd.nginx user
```

- d、重启 nginx 并再次访问统计页面

实验 3: 基于 IP 的身份验证（访问控制）

- a、接着上一个实验完成操作

b、在状态统计的 location 中添加：

```
allow 192.168.88.1;
deny 192.168.88.0/24;
仅允许 192.168.88.1 访问服务器
```

实验 4: nginx 的虚拟主机（基于域名）

- a、提前准备好两个网站的域名，并且规划好两个网站网页存放目录
- b、在 Nginx 主配置文件中并列编写两个 server 标签，并分别写好各自信息

```
server {
    listen 80;
    server_name blog.atguigu.com;
    index index.html index.htm index.php;
    root  html/blog;
    access_log  logs/blog-access.log  main;
}
server {
    listen 80;
    server_name bbs.atguigu.com;
    index index.html index.htm index.php;
    root  html/bbs;
    access_log  logs/bbs-access.log  main;
}
```

c、分别访问两个不同的域名验证结果

实验 5: nginx 的反向代理

代理和反向代理？

代理：找别人代替你去完成一件你完不成的事(代购)，代理的对象是客户端

反向代理：替厂家卖东西的人就叫反向代理(烟酒代理)，代理的对象是服务器端

- a、在另外一台机器上安装 apache，启动并填写测试页面
- b、在 nginx 服务器的配置文件中添加（**写在某一个网站的 server 标签内**）

```
location / {
    proxy_pass http://192.168.88.100:80;    #此处填写 apache 服务器的 IP 地址
}
```

c、重启 nginx，并使用客户端访问测试

实验 6: 负载均衡（负载均衡）

负载均衡（Load Balance）其意思就是将任务分摊到多个操作单元上进行执行，例如 Web 服务器、FTP 服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。

- a、使用默认的 rr 轮训算法，修改 nginx 配置文件

```
upstream bbs {    #此标签在 server 标签前添加
    server 192.168.88.100:80;
    server 192.168.88.200:80;
```

```
}
server {
    .....;
    #修改自带的 location / 的标签，将原内容删除，添加下列两项
    location / {
        proxy_pass http://bbs; #添加反向代理，代理地址填写 upstream 声明的名字
        proxy_set_header Host $host; #重写请求头部，保证网站所有页面都可访问成功
    }
}
```

c、开启并设置两台 88.100 & 88.200 的主机

安装 apache 并设置不同的 index.html 页面内容（设置不同页面是为了看实验效果）

d、重启 nginx，并使用客户端访问测试

拓展补充：rr 算法实现加权轮询（后期集群再讲更多算法类型和功能）

```
upstream bbs {
    server 192.168.88.100:80 weight=1;
    server 192.168.88.200:80 weight=2;
}
```

实验 7：nginx 实现 https {证书+rewrite}

a、安装 nginx 时，需要将--with-http_ssl_module 模块开启

b、在对应要进行加密的 server 标签中添加以下内容开启 SSL

```
server {
    .....;
    ssl on;
    ssl_certificate /usr/local/nginx/conf/ssl/atguigu.crt;
    ssl_certificate_key /usr/local/nginx/conf/ssl/atguigu.key;
    ssl_session_timeout 5m;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers
    "EECDH+CHACHA20:EECDH+CHACHA20-draft:EECDH+AES128:RSA+AES128:EECDH+AES2
56:RSA+AES256:EECDH+3DES:RSA+3DES:!MD5";
}
```

c、生成证书和密钥文件

注意：在实验环境中可以用命令生成测试，在生产环境中必须要在 https 证书厂商注册

```
# openssl genrsa -out atguigu.key 1024
```

建立服务器私钥，生成 RSA 密钥

```
# openssl req -new -key atguigu.key -out atguigu.csr
```

需要依次输入国家，地区，组织，email。最重要的是有一个 common name，可以写你的名字或者域名。如果为了 https 申请，这个必须和域名吻合，否则会引发浏览器警报。生成的 csr 文件交给 CA 签名后形成服务端自己的证书

```
# openssl x509 -req -days 365 -sha256 -in atguigu.csr -signkey atguigu.key -out atguigu.crt
```

生成签字证书

```
# cp atguigu.crt /usr/local/nginx/conf/ssl/atguigu.crt  
# cp atguigu.key /usr/local/nginx/conf/ssl/atguigu.key
```

将私钥和证书复制到指定位置

d、设置 http 自动跳转 https 功能
原有的 server 标签修改监听端口

```
server {  
    .....;  
    listen 443;  
}
```

新增以下 server 标签（利用虚拟主机+rewrite 的功能）

```
server{  
    listen 80;  
    server_name bbs.atguigu.com;  
    rewrite ^(.*)$ https://bbs.atguigu.com permanent;  
    root html;  
    index index.html index.htm;  
}
```

e、重启 nginx，并测试

Nginx 配置文件详解

```
user www www;
```

#程序运行用户和组

```
worker_processes auto;
```

#启动进程，指定 nginx 启动的工作进程数量，建议按照 cpu 数目来指定，一般等于 cpu 核心数目

```
error_log /home/wwwlogs/nginx_error.log crit;
```

#全局错误日志

```
pid /usr/local/nginx/logs/nginx.pid;
```

#主进程 PID 保存文件

```
worker_rlimit_nofile 51200;
```

#文件描述符数量

```
events
```

```
{
```

```
use epoll;
```

#使用 epoll 模型，对于 2.6 以上的内核，建议使用 epoll 模型以提高性能

```
worker_connections 51200;
```

#工作进程的最大连接数量，根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把 cpu 跑到 100% 就行每个进程允许的最多连接数，理论上每台 nginx 服务器的最大连接数为 $worker_processes * worker_connections$ ，具体还要看服务器的硬件、带宽等。

```
}
```

```
http
```

#整体环境配置--网站配置

```
{
```

```
include mime.types;
```

```
default_type application/octet-stream;
```

#设定 mime 类型,文件传送类型由 mime.type 文件定义

```
server_names_hash_bucket_size 128;
```

#保存服务器名字的 hash 表大小

```
client_header_buffer_size 32k;
```

#客户端请求头部缓冲区大小

```
large_client_header_buffers 4 32k;
```

#最大客户端头缓冲大小


```
client_max_body_size 50m;
```

```
#客户端最大上传文件大小（M）
```

```
sendfile on;
```

#sendfile 指令指定 nginx 是否调用 sendfile 函数来输出文件，对于普通应用，必须设为 on。如果用来进行下载等应用磁盘 IO 重负载应用，可设置为 off，以平衡磁盘与网络 I/O 处理速度，降低系统的 uptime。

```
tcp_nopush on;
```

#这个是默认的，结果就是数据包不会马上传送出去，等到数据包最大时，一次性的传输出去，这样有助于解决网络堵塞。（只在 sendfile on 时有效）

```
keepalive_timeout 60;
```

```
#连接超时时间
```

```
tcp_nodelay on;
```

```
#禁用 nagle 算法，也即不缓存数据。有效解决网络阻塞
```

```
fastcgi_connect_timeout 300;
```

```
fastcgi_send_timeout 300;
```

```
fastcgi_read_timeout 300;
```

```
fastcgi_buffer_size 64k;
```

```
fastcgi_buffers 4 64k;
```

```
fastcgi_busy_buffers_size 128k;
```

```
fastcgi_temp_file_write_size 256k;
```

```
#fastcgi 设置
```

```
gzip on;
```

```
gzip_min_length 1k;
```

```
gzip_buffers 4 16k;
```

```
gzip_http_version 1.1;
```

```
gzip_comp_level 2;
```

```
gzip_types text/plain application/javascript application/x-javascript text/javascript  
text/css application/xml application/xml+rss;
```

```
gzip_vary on;
```

```
gzip_proxied expired no-cache no-store private auth;
```

```
gzip_disable "MSIE [1-6]\\.";
```

```
#limit_conn_zone $binary_remote_addr zone=perip:10m;
```

```
##If enable limit_conn_zone,add "limit_conn perip 10;" to server section.
```

```
server_tokens off;
```

```
#隐藏 nginx 版本号（curl -I 192.168.4.154 可以查看，更加安全）
```

```
#log format
```

```
log_format access '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" $http_x_forwarded_for';
#定义日志格式
```

server

```
{
    listen 80 default_server;
    #listen [::]:80 default_server ipv6only=on;
    #监听 80 端口，WEB 服务的监听设置，可以采用"IP 地址:端口"形式
    server_name www.lnmp.org lnmp.org;
    #服务器名，可以写多个域名，用空格分隔
    index index.html index.htm index.php;
    #默认网页文件
    root /home/wwwroot/default;
    #网页主目录

    #error_page 404 /404.html;
    include enable-php.conf;

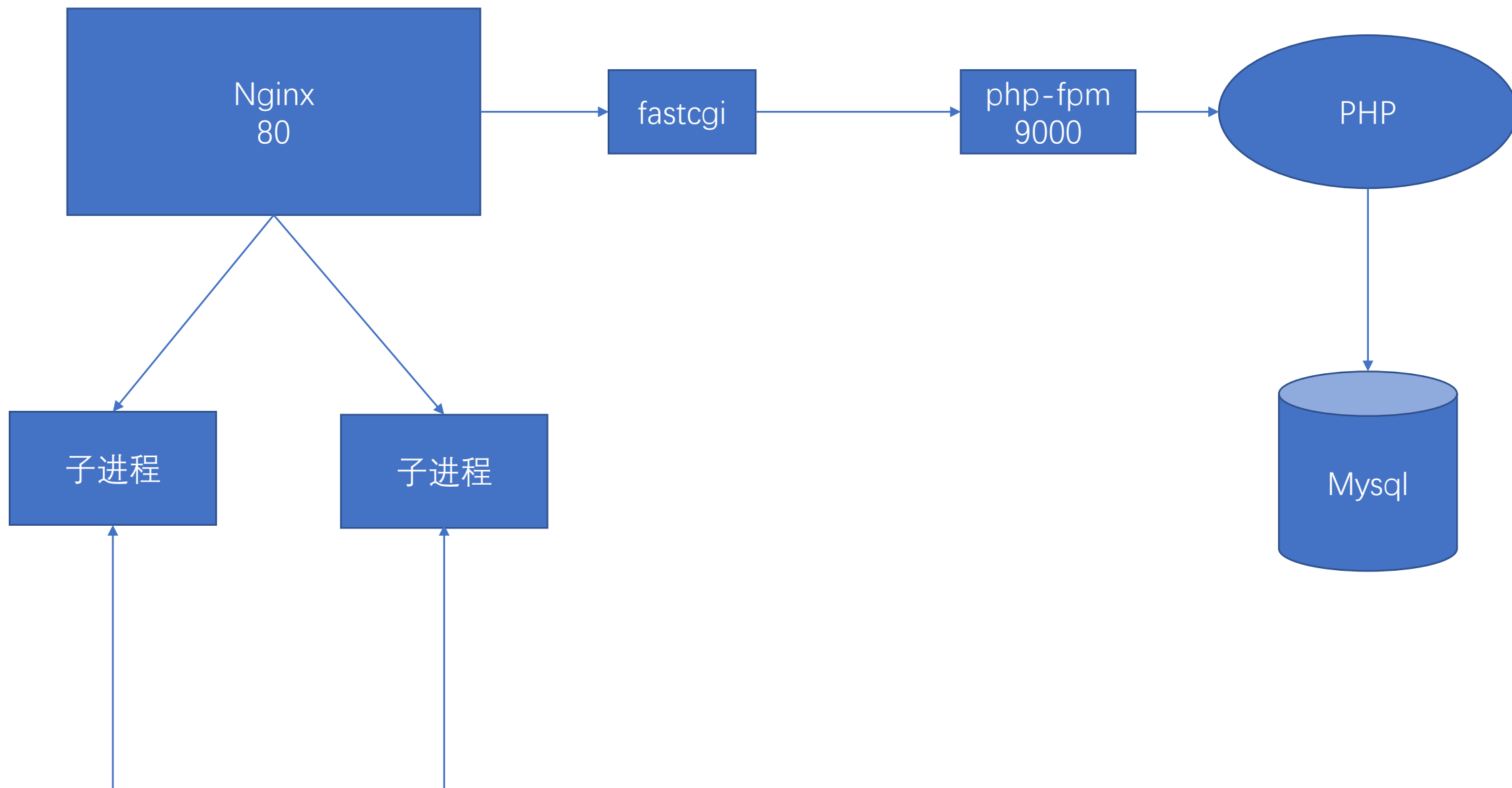
    location /nginx_status
    {
        stub_status on;
        access_log off;
    }
    #开启 status 状态监测
    location ~ .*\. (gif|jpg|jpeg|png|bmp|swf)$
    {
        expires 30d;
    }
    #静态文件处理，保存期 30 天
    location ~ .*\. (js|css)?$
    {
        expires 12h;
    }
    #js 和 css 文件处理，保存期 12 小时
    location ~ /\.
    {
        deny all;
    }

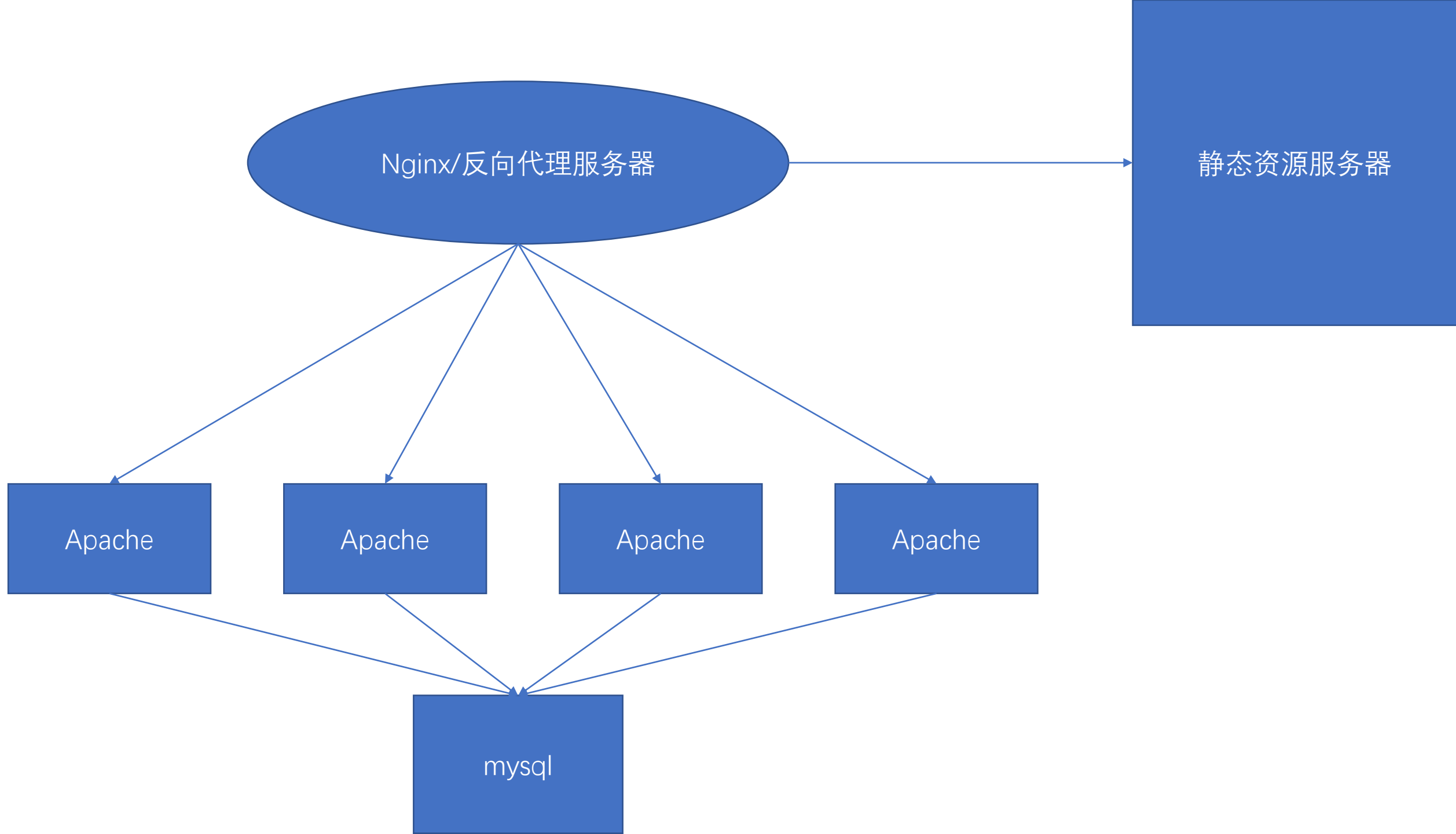
    access_log /home/wwwlogs/access.log access;
```

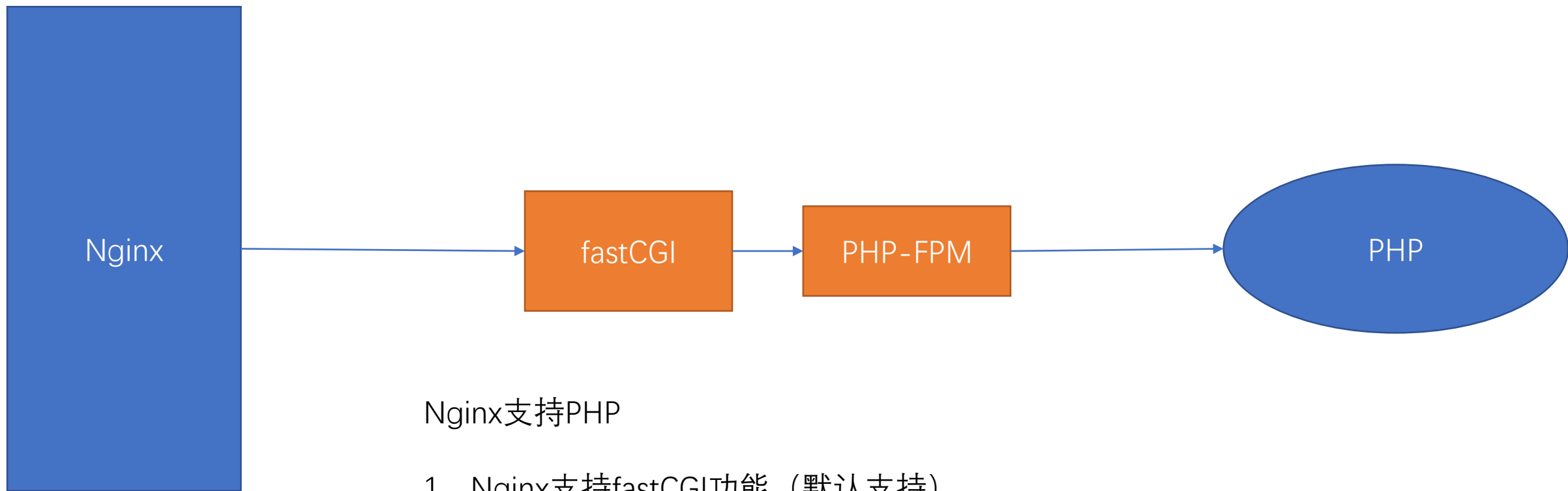
#正确访问日志

```
}  
include vhost/*.conf;  
#vhost/下子配置文件生效  
}
```

尚硅谷

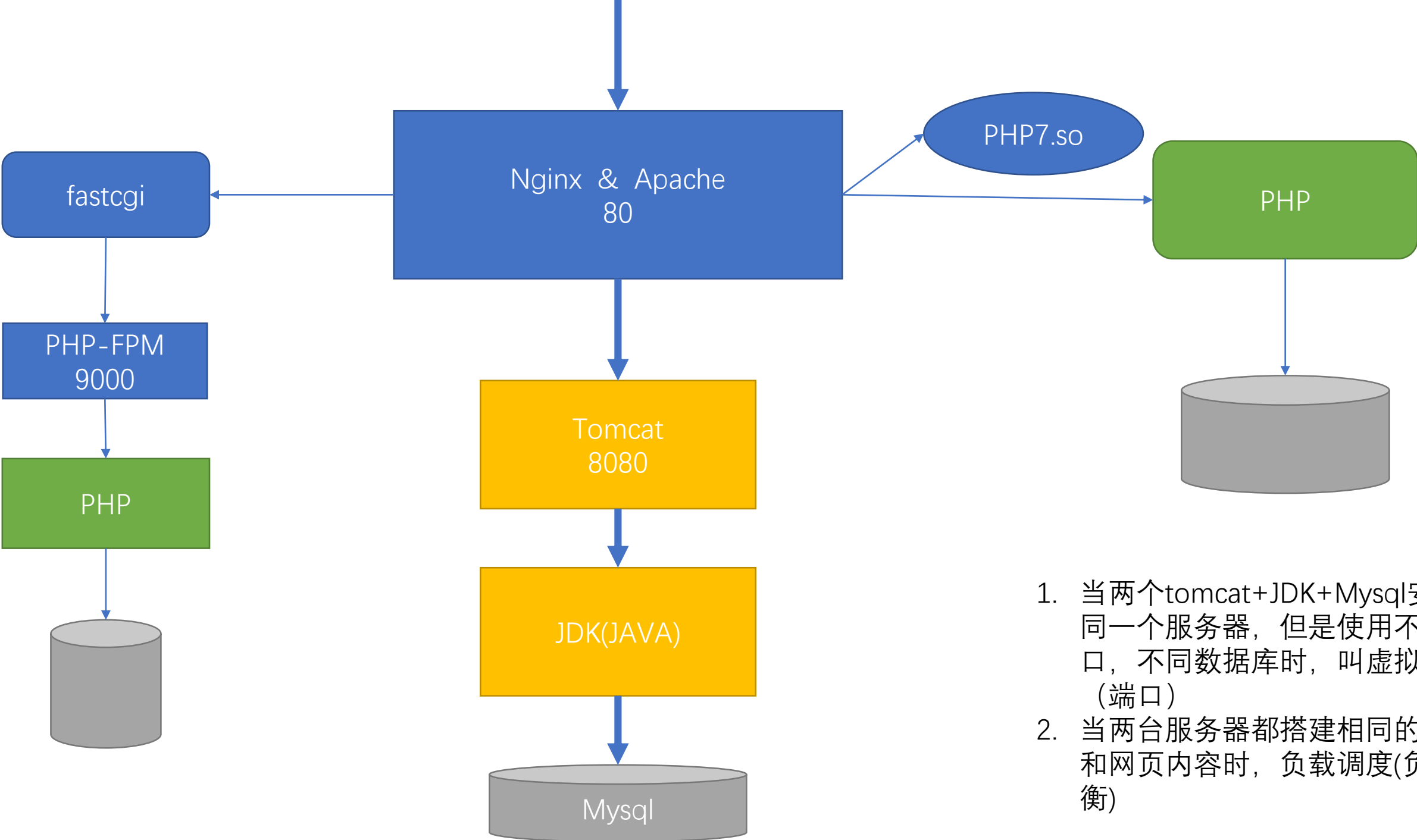






Nginx支持PHP

1. Nginx支持fastCGI功能（默认支持）
2. PHP编译时开启FPM服务（编译时指定）
3. 在Nginx配置文件中添加匹配规则（匹配后缀是.php）
（修改nginx.conf）



1. 当两个tomcat+JDK+Mysql安装在同一个服务器，但是使用不同端口，不同数据库时，叫虚拟主机（端口）
2. 当两台服务器都搭建相同的环境和网页内容时，负载调度(负载均衡)

