Apusic 金蝶天燕

生成时间：2026-05-19

性能优化手册-标准版

金蝶Apusic负载均衡器

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

- 1 一、前言
  - 1.1 面向读者
  - 1.2 版本更新说明
- 2 二、性能瓶颈来源分析（操作系统 + ALB双维度详解）
  - 2.1 2.1 操作系统层面瓶颈
    - 2.1.1 （1）文件描述符（FD）限制
      - 2.1.1.1 为什么存在?
      - 2.1.1.2 如何修改?
      - 2.1.1.3 如何验证?
    - 2.1.2 （2）TCP 连接队列与网络栈参数
      - 2.1.2.1 为什么存在?
      - 2.1.2.2 如何修改?
      - 2.1.2.3 如何验证?
    - 2.1.3 3）TIME_WAIT 连接堆积
      - 2.1.3.1 为什么存在?
      - 2.1.3.2 如何缓解?
      - 2.1.3.3 如何验证?
    - 2.1.4 4）CPU 与中断亲和性
      - 2.1.4.1 为什么存在?
      - 2.1.4.2 如何优化?
      - 2.1.4.3 如何验证?
  - 2.2 12.2 ALB 配置瓶颈
    - 2.2.1 （1）Worker 进程与连接数配置不当
      - 2.2.1.1 为什么存在?
      - 2.2.1.2 如何优化?
      - 2.2.1.3 如何验证?
    - 2.2.2 2）缓冲区（Buffer）配置不合理
      - 2.2.2.1 为什么存在?
      - 2.2.2.2 如何优化?
      - 2.2.2.3 如何验证?
    - 2.2.3 3）日志写入频繁导致 I/O 瓶颈
      - 2.2.3.1 为什么存在?

- 2.2.3.2 如何验证?
- 2.2.4 （4）SSL/TLS 握手开销大
- 2.2.4.1 为什么存在?
- 2.2.4.2 如何优化?
- 2.2.4.3 如何验证?
- 2.2.5 （5）静态资源未启用高效传输机制
- 2.2.5.1 为什么存在?
- 2.2.5.2 如何优化?
- 2.2.5.3 如何验证?
- 2.3 2.3 综合验证方法
- 2.4 3.4 小结
- 3 三、超时控制与长连接优化（客户端 + 上游）
- 3.1 3.1、超时设置不当引发的性能瓶颈
- 3.2 3.2 客户端相关超时（Client-side Timeouts）
- 3.2.1 ✅ 优化建议：
- 3.2.2 🔍 验证方式：
- 3.3 3.3 上游服务相关超时（Upstream Timeouts）
- 3.3.1 ✅ 优化建议（以 API 网关为例）：
- 3.3.2 🔍 验证方式：
- 4 四、长连接配置指南
- 4.1 4.1 客户端 ↔ ALB：启用 HTTP Keep-Alive
- 4.2 4.2 ALB ↔ 上游：启用 Upstream Keep-Alive（关键！）
- 4.3 4.3 后端服务：需支持 HTTP/1.1 并保持连接
- 4.3.1 4.4 综合验证：如何确认长连接与超时配置生效?
- 5 五、典型场景优化案例
- 5.1 5.1 场景一： Web网站访问加速优化
- 5.1.1 业务场景描述
- 5.1.2 优化目标
- 5.1.3 优化方案
- 5.1.4 优化效果
- 5.2 5.2 场景二： API网关性能优化
- 5.2.1 业务场景描述
- 5.2.2 优化目标
- 5.2.3 优化方案

- 5.3 5.3 场景三：文件下载服务优化
  - 5.3.1 业务场景描述
  - 5.3.2 优化目标
  - 5.3.3 优化方案
  - 5.3.4 优化效果
- 5.4 5.4 场景四：ALB反向代理AAS服务性能优化
  - 5.4.1 1. 操作系统层面调优（Linux）
  - 5.4.2 2. ALB 配置优化
  - 5.4.3 3. 压测验证
- 5.5 5.5 场景五：ALB 百万级并发连接测试（C1000K）
  - 5.5.1 背景
    - 5.5.1.1 ✅ 典型业务场景:
    - 5.5.1.2 ⚠️ 重要前提：
    - 5.5.1.3 ❗ 核心难点：
  - 5.5.2 解决方案
  - 5.5.3 操作系统全面调优（Linux）
    - 5.5.3.1 1. 文件描述符（FD）——最关键！
    - 5.5.3.2 2. 网络栈调优
    - 5.5.3.3 3. 内核与硬件建议
  - 5.5.4 ALB配置优化
  - 5.5.5 后端优化
  - 5.5.6 常见问题与规避
  - 5.5.7 总结
- 6 六、性能测试与问题排查
  - 6.1 6.1 诊断命令汇总
  - 6.2 6.2 错误现象：压测时大量请求返回 502 Bad Gateway
    - 6.2.1 🔍 错误分析
    - 6.2.2 ⚙️ 解决方案
    - 6.2.3 ✅ 验证方法
  - 6.3 6.3 错误现象：压测初期正常，随后 QPS 骤降，连接堆积
    - 6.3.1 🔍 错误分析
    - 6.3.2 ⚙️ 解决方案
    - 6.3.3 ✅ 验证方法
  - 6.4 6.4. 错误现象：压测时出现大量 504 Gateway Timeout

- 6.4.1 🔍 错误分析
- 6.4.2 ⚙️ 解决方案
- 6.4.3 ✅ 验证方法
- 6.5 6.5 错误现象：压测时客户端报 “Connection reset by peer” 或 “Connection closed”
  - 6.5.1 🔍 错误分析
  - 6.5.2 ⚙️ 解决方案
  - 6.5.3 ✅ 验证方法
- 6.6 6.6 错误现象：启用 upstream keepalive 后，QPS 无提升甚至下降
  - 6.6.1 🔍 错误分析
  - 6.6.2 ⚙️ 解决方案
- 6.7 6.7 错误现象：压测时系统 CPU sys%（内核态）过高，达 70%+
  - 6.7.1 🔍 错误分析
  - 6.7.2 ⚙️ 解决方案
  - 6.7.3 ✅ 验证方法
- 6.8 6.8 错误现象：多核 CPU 下，仅 CPU0 负载 100%，其他核空闲
  - 6.8.1 🔍 错误分析
  - 6.8.2 ⚙️ 解决方案
  - 6.8.3 ✅ 验证方法
- 7 七、性能测试错误速查表
- 8 八、性能优化关键参数清单
  - 8.1 8.1 全局与进程控制
  - 8.2 8.2 事件模型（Events）
  - 8.3 8.3 HTTP 全局配置
  - 8.4 8.4 客户端连接与超时
  - 8.5 8.5 反向代理（Proxy）
  - 8.6 8.6 Upstream 长连接池
  - 8.7 8.7 日志

1 一、前言

本文档旨在为金蝶Apusic负载均衡器V2.0.5标准版用户提供全面、系统的性能优化指导。通过深入分析操作系统层面和ALB自身配置两个维度的性能瓶颈，结合典型业务场景的优化案例，帮助用户解决在高并发环境下可能遇到的性能问题。

文档内容涵盖了从基础的系统参数调优到复杂的百万级并发连接优化，从常见的超时配置到关键的长连接复用策略，以及在性能测试过程中可能遇到的各种问题排查方法。通过本文的指导，用户可以：

- 识别和解决ALB在高并发场景下的性能瓶颈
- 合理配置操作系统和ALB参数以提升系统整体性能
- 掌握长连接优化策略，有效降低系统资源消耗
- 快速定位和解决性能测试中遇到的常见问题
- 根据具体业务场景实施相应的优化方案

1.1 面向读者

本文档主要面向金蝶Apusic负载均衡器的开发人员、运维人员及相关管理人员，适用于Linux系统环境（包括Kylin、UOS、OpenEuler、CentOS、Ubuntu等发行版）。通过系统地学习和应用本文档中的优化建议，用户可以显著提升ALB的性能表现，确保系统在高负载情况下依然保持稳定、高效的服务能力。

1.2 版本更新说明

本文根据实际情况进行更新，最新版本包含历史修改记录。

日期	手册版本	适用产品	更新说明
2025年11月	V1.0	ALB V2.0.5 标准版	性能优化说明、优化案例、性能错误排查、性能相关指令速查

2 二、性能瓶颈来源分析（操作系统 + ALB双维度详解）

ALB虽然以高性能著称，但在高并发、大流量或复杂业务场景下，其实际表现往往受限于两个关键层面：操作系统资源调度策略 和 ALB 自身配置合理性。这两者相互影响，需协同调优。

总体原则：

> *操作系统提供"舞台"（CPU、内存、网络、文件系统等资源）；*

> *ALB 是"演员"，其表演效果取决于舞台条件与自身编排（配置）。*

> *忽略任一层面，都可能导致性能无法达到预期。*

2.1 2.1 操作系统层面瓶颈

2.1.1 （1）文件描述符（FD）限制

2.1.1.1 为什么存在?

Linux 默认限制每个进程可打开的文件描述符数量（通常为 1024），防止资源耗尽。而每个 TCP 连接、静态文件句柄均占用一个 FD。

当并发连接数超过限制，ALB会拒绝新连接，并在`error log`中记录`Too many open files`。

2.1.1.2 如何修改?

```
# 编辑alb启动脚本，脚本开头添加以下两行内容
# 【ALB安装目录】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 65536
ulimit -Hn 65536

# 若使用 systemd，则在[Service]结尾添加LimitNOFILE=65536
# 编辑 /etc/systemd/system/alb.service
[Service]
....
User=root
Type=forking
LimitNOFILE=65536   # 添加这一行内容
# 修改完后，使用systemctl daemon-reload 然后重启ALB
```

2.1.1.3 如何验证?

```
# 查看 worker 进程 FD 限制
cat /proc/$(pgrep alb | tail -1)/limits | grep "open files"

# 实时 FD 使用量
lsof -p $(pgrep alb | tail -1) | wc -l
```

2.1.2 （2）TCP 连接队列与网络栈参数

2.1.2.1 为什么存在?

内核通过 somaxconn（accept 队列）和 tcp_max_syn_backlog（SYN 队列）限制未完成连接的数量。默认值（如 128）在高并发场景下极易成为瓶颈，导致新连接被丢弃，表现为客户端连接超时。

2.1.2.2 如何修改?

```
# 编辑 /etc/sysctl.conf
net.core.somaxconn = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.tcp_abort_on_overflow = 1  # 可选：队列满时立即 RST，便于发现问题
```

执行 `sysctl -p` 生效。

同时在 ALB配置文件中alb.conf针对监听端口的配置添加backlog 配置：

```
listen 80 backlog=65535;
```

2.1.2.3 如何验证?

```
ss -lnt  # 查看 Recv-Q（当前 accept 队列长度），若持续接近 backlog 值则需扩容
```

2.1.3 3）TIME_WAIT 连接堆积

2.1.3.1 为什么存在?

主动关闭连接的一方进入 TIME_WAIT 状态（约 60 秒），确保网络中残余数据包不会干扰新连接。在短连接高频请求场景（如移动端 API），大量 TIME_WAIT 会耗尽本地端口（ephemeral port range），导致 `Cannot assign`

`requested address`。

2.1.3.2 如何缓解?

```
# 编辑/etc/sysctl.conf
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_fin_timeout = 15
```

> *修改完使用 sysctl -p 生效。 ⚠ 禁用 `tcp_tw_recycle`（已废弃且 NAT 下不安全）。*

2.1.3.3 如何验证?

```
# 编辑
ss -ant | awk '/TIME-WAIT/ {count++} END {print count+0}'
ss -s  # 查看 total connections 和 timewait 数量
```

2.1.4 4）CPU 与中断亲和性

2.1.4.1 为什么存在?

默认网卡中断集中在 CPU0，导致单核过载，而其他核心空闲。alb多 worker 无法均衡利用多核。

2.1.4.2 如何优化?

启用 `reuseport`（推荐）：

```
listen 80 reuseport;
```

内核自动将新连接分发到各 worker 的独立 socket，实现负载均衡。 或手动绑定 IRQ 到不同 CPU（高级）。

2.1.4.3 如何验证?

```
top  # 观察各 CPU 核心使用率是否均衡
```

2.2 12.2 ALB 配置瓶颈

2.2.1 （1）Worker 进程与连接数配置不当

2.2.1.1 为什么存在?

- worker_processes 设置过少（如 1），无法利用多核；
- worker_connections 设置过低，单 worker 无法处理足够并发；
- 未设置 worker_rlimit_nofile，导致即使系统允许，ALB 仍受内部限制。

2.2.1.2 如何优化?

```
worker_processes auto;              # 自动匹配 CPU 核数
worker_rlimit_nofile 65536;         # 提升内部 FD 限制

events {
    use epoll;                      # Linux 必须启用
    worker_connections 65536;       # 单 worker 最大连接数
    multi_accept on;                # 一次 accept 多个连接
}
```

2.2.1.3 如何验证?

- 理论最大并发 = worker_processes × worker_connections
- 结合压测观察是否达到预期 QPS

2.2.2 2）缓冲区（Buffer）配置不合理

2.2.2.1 为什么存在?

- proxy_buffering off 会导致后端响应直接流式转发，增加 client 与 backend 的耦合，降低吞吐；
- Buffer 太小（如默认 4k/8k）在大响应体场景下频繁 syscall，增加 CPU 开销；
- Buffer 太大会浪费内存。

2.2.2.2 如何优化?

```
proxy_buffering on;
proxy_buffer_size 16k;
proxy_buffers 8 32k;          # 总缓冲 ≈ 256k，按业务调整
proxy_busy_buffers_size 64k;
```

2.2.2.3 如何验证?

- 对比开启/关闭 buffering 的 QPS 与延迟；
- 使用 strace -p 观察 read/write 系统调用频率。

2.2.3 3）日志写入频繁导致 I/O 瓶颈

2.2.3.1 为什么存在?

默认每请求写一行 access log，高并发下磁盘 I/O 成为瓶颈，尤其在机械硬盘或云盘 IOPS 有限时。 如何优化?

```
# 方案1：关闭非必要日志（生产慎用）
access_log off;

# 方案2：启用缓冲写入
access_log logs/access.log main buffer=64k flush=5s;
```

2.2.3.2 如何验证?

```
iostat -x 1  # 观察 %util 和 await 是否下降
```

2.2.4 （4）SSL/TLS 握手开销大

2.2.4.1 为什么存在?

每次 HTTPS 新连接需完整 TLS 握手（2-RTT），计算密集（尤其 RSA 密钥交换），消耗大量 CPU。

2.2.4.2 如何优化?

```
ssl_session_cache shared:SSL:10m;      # 多 worker 共享 session
ssl_session_timeout 10m;
ssl_session_tickets on;                # 启用 ticket 快速恢复（注意 key
轮换）
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off;         # 让客户端优先选择高效 cipher
keepalive_timeout 75s;                 # 复用 TCP+TLS 连接
```

2.2.4.3 如何验证?

- 使用 openssl s_client -connect host:443 -reconnect 测试 session 复用;
- 压测对比 HTTPS 与 HTTP 的 QPS 差距是否缩小;
- 监控 CPU 使用率（top 中 %sys 是否过高）。

2.2.5 （5）静态资源未启用高效传输机制

2.2.5.1 为什么存在?

未开启 sendfile、tcp_nopush 等，导致内核多次拷贝数据，增加延迟。

2.2.5.2 如何优化?

```
sendfile on;            # 零拷贝传输文件
tcp_nopush on;          # 累积数据后再发送，提升吞吐
tcp_nodelay on;         # 小包立即发送（适用于动态内容）
open_file_cache max=10000 inactive=60s;   # 缓存文件元数据
```

2.2.5.3 如何验证?

- 对比开启前后静态文件下载速度;
- 使用 perf 或 bpftrace 观察 page fault 和 copy 次数。

2.3 2.3 综合验证方法

完成上述任一优化后，必须通过以下方式验证效果:

验证目标	工具/命令	关注指标
并发连接能力	`wrk -c 10000 -t 10 -d 30s URL`	Requests/sec, Latency
连接队列状态	`ss -lnt`	Recv-Q 是否接近 backlog
FD 使用情况	`lsof -p PID | wc -l`	
CPU 利用均衡性	`top` 或 `htop`	各核负载是否均匀
网络错误	`netstat -s | grep -i drop`	
ALB 内部状态	启用 `stub_status`	active/accepts/handled/requests
磁盘 I/O 压力	`iostat -x 1`	%util, await

2.4 3.4 小结

维度	典型瓶颈	优化方向
操作系统	FD 限制、TCP 队列、TIME_WAIT	调整 limits、sysctl 参数
ALB 配置	Worker 数、Buffer、日志、SSL	合理设置 events/http 模块

3 三、超时控制与长连接优化（客户端 + 上游）

3.1 3.1、超时设置不当引发的性能瓶颈

ALB 涉及多个超时参数，分别作用于 **客户端 ↔ ALB** 和 **ALB ↔ 上游服务** 两个方向。若配置不合理，会导致：

- 连接无法及时释放 → `worker_connections` 耗尽；
- 请求卡住不返回 → 客户端超时、用户体验差；
- 后端异常时 ALB 积压大量 pending 请求 → 内存/CPU 耗尽。

3.2 3.2 客户端相关超时（Client-side Timeouts）

参数	默认值	作用	风险
`client_header_timeout`	60s	读取客户端请求头的超时	慢客户端攻击（Slowloris）可耗尽连接
`client_body_timeout`	60s	读取客户端请求体的超时	大文件上传未限速时卡住 worker
`keepalive_timeout`	75s	客户端 keep-alive 连接保持时间	设置过大会导致 idle 连接堆积

3.2.1 ✅ 优化建议：

```
# 根据业务调整，API 场景可缩短
client_header_timeout 10s;
client_body_timeout 10s;

# 合理设置 keepalive，兼顾复用与资源回收
keepalive_timeout 30s;          # 保持 30 秒
keepalive_requests 1000;        # 单连接最多处理 1000 个请求（防内存泄漏）
```

3.2.2 🔍 验证方式：

- 模拟慢请求（如使用curl限制速度发送大文件slow.txt[10MB]： `curl -H "Transfer-Encoding: chunked" --limit-rate 100 --data-binary @slow.txt -X POST http://ip:port/api` ）；
- 观察 error log 是否出现 `client timed out` ；

- 使用 `ss -o state established` 查看连接 ESTAB 时间是否超过预期。

3.3 3.3 上游服务相关超时（Upstream Timeouts）

参数	默认值	作用	风险
`proxy_connect_timeout`	60s	与 upstream 建立 TCP 连接的超时	后端宕机时阻塞 worker
`proxy_send_timeout`	60s	向 upstream 发送请求的超时	网络拥塞或后端接收慢
`proxy_read_timeout`	60s	从 upstream 读取响应的超时	后端处理慢或 hang 死

> ⚠️ *默认 60s 在高并发场景下极其危险！一个慢接口可能导致数千连接被占用。*

3.3.1 ✅ 优化建议（以 API 网关为例）：

```
location /api/ {
    proxy_pass http://backend;

    # 连接超时：通常 1~3 秒足够，高并发大压力下 可适当调大
    proxy_connect_timeout 10s;

    # 发送/读取超时：根据业务 P99 响应时间设定（如 500ms ~ 5s）但在 高并发或大文件上传场景下应设得长一些，比如120s
    proxy_send_timeout 60s;
    proxy_read_timeout 60s;

    # 其他必要配置
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
```

3.3.2 🔍 验证方式：

- 故意让后端 sleep 10s，观察 ALB 是否在 5s 后返回 504 Gateway Timeout；
- 监控 alb_status 中 active connections 是否在异常时飙升；
- 日志中应出现 `upstream timed out (110: Connection timed out)` 。

> 💡 *最佳实践：*

> *超时时间应 略大于后端 P99 响应时间，并配合熔断/降级机制*

4 四、长连接配置指南

长连接需在 三个环节 协同配置才能生效：

```
[客户端] ←(HTTP Keep-Alive)→ [ALB] ←(Upstream Keep-Alive)→ [上游服务]
```

4.1 4.1 客户端 ↔ ALB：启用 HTTP Keep-Alive

ALB 默认支持客户端 keep-alive，但需合理配置：

```
http {
    # 全局 keepalive 设置
    keepalive_timeout 30s;        # 连接保持 30 秒
    keepalive_requests 1000;      # 单连接最多处理 1000 个请求（防内存泄漏）

    # 对 HTTP/2 自动启用多路复用，无需额外配置
}
```

> ✅ *适用于 Web 浏览器、移动端 App、内部服务调用等支持 keep-alive 的客户端。*

4.2 4.2 ALB ↔ 上游：启用 Upstream Keep-Alive（关键！）

已在前文详述，此处整合为完整模板：

```
upstream backend {
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
    keepalive 64;  # 每个 worker 缓存 64 个空闲连接
}

server {
    location /api/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
```

```
        proxy_set_header Connection "";  # 清除 Connection 头

        # 超时控制（必须配合长连接）
        proxy_connect_timeout 3s;
        proxy_send_timeout 120s;
        proxy_read_timeout 120s;
    }
}
```

4.3 4.3 后端服务：需支持 HTTP/1.1 并保持连接

- 后端框架（如 Spring Boot、Gin、Flask）默认通常支持；
- 确保未主动发送 Connection: close；
- 若使用 TLS，建议启用 session resumption。

4.3.1 4.4 综合验证：如何确认长连接与超时配置生效?

验证目标	方法
客户端 keep-alive	用浏览器开发者工具或 `curl -v` 查看是否复用连接；或用 `tcpdump` 观察同一 TCP 流中多个 HTTP 请求
upstream keep-alive	`ss -tnp | grep alb | grep :8080` 查看 ESTAB 连接是否稳定存在
超时生效	模拟后端延迟 > timeout，确认 ALB 返回 504 而非 hang 住
连接复用效果	压测对比开启/关闭 keepalive 的 QPS 与 CPU 使用率

示例压测命令（验证 upstream keepalive）：

```
# 高并发短请求
wrk -t16 -c1000 -d30s --latency http://alb/api/fast-endpoint

# 观察指标：
# - Requests/sec 提升
# - ss -s 中 timewait 数量显著下降
# - 后端 accept() 次数减少
```

5 五、典型场景优化案例

5.1 5.1 场景一： Web网站访问加速优化

5.1.1 业务场景描述

某企业部署了一个门户网站，包含大量静态资源（图片、CSS、JS文件），访问量逐渐增加，用户反映页面加载较慢。

5.1.2 优化目标

提升静态资源访问速度，降低服务器负载，改善用户体验。

5.1.3 优化方案

- 启用静态资源缓存

```
http {
    # 开启文件缓存
    open_file_cache max=1000 inactive=20s;
    open_file_cache_valid 30s;
    open_file_cache_min_uses 2;
    open_file_cache_errors on;

    server {
        listen 80;
        root /var/www/html;

        # 针对静态资源设置缓存
        location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
            expires 1y;
            add_header Cache-Control "public, immutable";
        }
    }
}
```

- 启用gzip压缩

```
http {
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css application/json
application/javascript text/xml application/xml;
}
```

5.1.4 优化效果

- 静态资源加载速度提升50%以上
- 服务器CPU使用率降低20%
- 用户页面加载时间显著缩短

5.2 5.2 场景二： API网关性能优化

5.2.1 业务场景描述

某公司API网关需要处理大量移动端请求，高峰期出现响应慢、部分请求失败的情况。

5.2.2 优化目标

提升API网关并发处理能力，保证服务稳定性。

5.2.3 优化方案

- 1. 调整系统文件描述符限制

```
    # 编辑alb启动脚本，脚本开头添加以下两行内容
    # 【ALB安装目录】/bin/start-alb.sh或start-alb-and-console.sh
    ulimit -n 65536
    ulimit -Hn 65536
```

- 2. 优化ALB配置

```
worker_processes auto;
worker_rlimit_nofile 65536;
```

```
events {
    use epoll;
    worker_connections 4096;
    multi_accept on;
}

http {
    # 启用长连接复用
    upstream api_backend {
        server 192.168.1.10:8080 max_fails=3 fail_timeout=10s;
        server 192.168.1.11:8080 max_fails=3 fail_timeout=10s;
        keepalive 32;
    }
    # 限制单IP请求频率（防止恶意刷接口）
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

    server {
        listen 80 reuseport;

        # API路由
        location /api/ {
            proxy_pass http://api_backend;
            # 启用 HTTP/1.1 长连接
            proxy_http_version 1.1;
            proxy_set_header Connection "";

            # 合理设置超时
            proxy_connect_timeout 5s;
            proxy_send_timeout 120s;
            proxy_read_timeout 120s;
        }
```

```
        location /api/ {
            limit_req zone=api burst=20 nodelay;
        }
    }
}
```

5.3 5.3 场景三：文件下载服务优化

5.3.1 业务场景描述

企业需要提供大文件下载服务，用户反映下载速度慢，服务器资源占用高。

5.3.2 优化目标

提升大文件下载速度，降低服务器资源消耗。

5.3.3 优化方案

- 1. 启用大文件优化配置

```
http {
    # 调整缓冲区大小
    proxy_buffering on;
    proxy_buffer_size 128k;
    proxy_buffers 4 256k;
    proxy_busy_buffers_size 256k;

    # 启用分片下载支持
    proxy_http_version 1.1;

    server {
        listen 80;

        location /downloads/ {
            root /data;
```

```
            # 支持断点续传
            location ~ \.iso$ {
                add_header Accept-Ranges bytes;
            }
        }

        # 大文件下载专用配置
        location ~* \.(iso|zip|exe)$ {
            # 限制下载速度（防止带宽被占满）
            limit_rate 2m;   # 限制为2MB/s

            # 增加超时时间
            proxy_connect_timeout 60s;
            proxy_send_timeout 300s;
            proxy_read_timeout 300s;
        }
    }
}
```

5.3.4 优化效果

- 大文件下载速度提升60%
- 服务器内存占用降低30%
- 支持断点续传，改善用户体验

5.4 5.4 场景四：ALB反向代理AAS服务性能优化

> *目标：通过操作系统调优 + ALB 配置优化，使 ALB 能高效代理后端 AAS 服务，支撑高并发 API 请求。*

5.4.1 1. 操作系统层面调优（Linux）

提升文件描述限制

```
# 编辑alb启动脚本，脚本开头添加以下两行内容
# 【ALB安装目录】/bin/start-alb.sh或start-alb-and-console.sh
```

```
ulimit -n 100000
ulimit -Hn 100000
```

优化TCP网络栈

```
# 编辑/etc/sysctl.conf，添加以下内容：
net.core.somaxconn = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 15
fs.file-max = 2097152

# 生效
sysctl -p
```

AAS性能优化： 略，确保开启长连接。

5.4.2 2. ALB 配置优化

```
user root;
worker_processes auto;
worker_rlimit_nofile 100000;

events {
    use epoll;
    worker_connections 65536;
    multi_accept on;
}

http {
    # 全局超时控制
    client_header_timeout 120s;
    client_body_timeout 120s;
    keepalive_timeout 300s;
    keepalive_requests 1000;
```

```
    upstream aas_backend {
        server 192.168.1.100:8080 max_fails=3 fail_timeout=10s;
        server 192.168.1.101:8080 max_fails=3 fail_timeout=10s;
        server 192.168.1.102:8080 max_fails=3 fail_timeout=10s;
        keepalive 128;              # 每个 worker 保持 128 个长连接
        keepalive_requests 1000;
        keepalive_timeout 75s;
    }

    server {
        listen 8080 reuseport;

        location / {
            proxy_pass http://aas_backend/;

            # 启用与 AAS 的长连接
            proxy_http_version 1.1;
            proxy_set_header Connection "";

            # 超时设置（略大于 AAS 处理时间）
            proxy_connect_timeout 10s;
            proxy_send_timeout 120s;
            proxy_read_timeout 120s;

            # 传递客户端信息
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;

        }

        # 日志缓冲写入
        access_log access.log main buffer=64k flush=5s;
```

```
    }
}
```

5.4.3 3. 压测验证

```
# 使用 wrk 模拟高并发请求
wrk -t16 -c2000 -d60s --latency http://your-alb-server:8080/api/test

# 观察指标：
# - Requests/sec 是否稳定提升
# - ALB error log 无 "too many open files"
# - ss -tnp | grep :8080 查看是否维持 ESTAB 连接（非大量 TIME-WAIT）
```

5.5 5.5 场景五：ALB 百万级并发连接测试（C1000K）

5.5.1 背景

5.5.1.1 ✅ 典型业务场景：

- WebSocket 长连接网关（如聊天、IoT 设备接入）
- HTTP 长轮询（Long Polling）服务
- Server-Sent Events (SSE) 推送平台
- 微服务 API 网关（高保活连接）

5.5.1.2 ⚠️ 重要前提：

百万并发 ≠ 百万 QPS。此处指 同时维持 1,000,000 个 TCP 连接，每个连接可能低频通信（如每分钟一次心跳），但必须保持 alive。

5.5.1.3 ❗ 核心难点：

ALB 需为每个连接维护两个 socket：client ↔ alb + alb ↔ backend → 总 FD 数 ≈ 2 × 并发连接数 若未启用 upstream keepalive，后端需承受百万级连接压力（几乎不可行） 必须通过 连接复用（长连接池） 将后端连接数压缩到合理范围

5.5.2 解决方案

- ALB 层：承担百万连接接入、TLS 终止、路由分发
- Backend 层：仅需处理 活跃请求，通过 小规模长连接池 与 ALB 通信
- 关键策略：前端海量连接 + 后端有限连接池

> 💡 *理想比例：*

> *1,000,000 客户端连接 → ALB → 1,000 ~ 10,000 个 upstream 长连接（复用率 100:1 ~ 1000:1）*

5.5.3 操作系统全面调优（Linux）

5.5.3.1 1. 文件描述符（FD）——最关键！

```
# 编辑alb启动脚本，脚本开头添加以下两行内容
# 【ALB安装目录】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 150000
ulimit -Hn 150000


# 全局文件句柄上限
echo 'fs.file-max = 2500000' >> /etc/sysctl.conf
echo 'fs.nr_open = 2500000' >> /etc/sysctl.conf
```

> 📌 *ulimit设置15W，全局设置250W?*
>
> > `ALB百万链接最低连接数`： *百万连接 × 2（client + upstream） = 200 万。*
> >
> > `ALB worker数`： *ulimit设置的是每个进程打开数量为15W，那么200万/15万 = 14 worker进程数（即建议16核及以上CPU）。*
> >
> > *设置250W： 除ALB需要用到的200W文件数外，系统自身和其他服务也需要打开文件，即250W打开文件数是整个系统文件句柄数的上限。*

5.5.3.2 2. 网络栈调优

```
# /etc/sysctl.conf
net.core.somaxconn = 1000000
net.core.netdev_max_backlog = 50000
net.ipv4.tcp_max_syn_backlog = 300000
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_synack_retries = 2

# TIME-WAIT 优化（ALB 主动关闭 client 连接时产生）
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 10
```

```
net.ipv4.tcp_max_tw_buckets = 2000000

# 内存缓冲区（支持大并发）
net.core.rmem_default = 262144
net.core.wmem_default = 262144
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 16384 67108864
net.ipv4.tcp_wmem = 4096 16384 67108864

# 端口范围（若 ALB 也作为客户端发起连接）
net.ipv4.ip_local_port_range = 1024 65535
```

修改完1和2，执行下面命令生效：

```
sysctl -p
```

5.5.3.3 3. 内核与硬件建议

- Linux 内核 ≥ 4.5（稳定支持 reuseport + epoll）
- CPU ≥ 16 核（百万连接上下文切换开销大）
- 内存 ≥ 64GB（每个连接约 4~8KB，百万连接约 8GB，加上 buffer 和后端缓存）
- 关闭 swap：`swapoff -a`
- 使用 SSD：减少日志或临时文件 I/O 影响

5.5.4 ALB配置优化

```
user root;
worker_processes auto;                  # 匹配 CPU 核数（如 16）
worker_rlimit_nofile 150000;           # 单个worker打开文件数，和连接数匹配

error_log error.log warn;

events {
    use epoll;
    worker_connections 150000;          # 单 worker 支持 15 万连接
 (16CPU×15w=240w)
```

```
    multi_accept on;
}

http {
    # 关闭 access log（压测阶段），或使用异步缓冲
    access_log off;
    error_log error.log crit;

    # 高效传输
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;

    # 客户端 keep-alive（维持长连接）
    keepalive_timeout 300s;           # 5 分钟无操作断开
    keepalive_requests 1000000;       # 防止单连接内存泄漏

    # Upstream 长连接池（核心！）
    upstream backend_pool {
        # 假设 10 台后端，每台分配 1000 长连接 → 总 10,000
        server 10.0.1.10:8080 max_conns=1000 max_fails=3
fail_timeout=10s;;
        server 10.0.1.11:8080 max_conns=1000 max_fails=3
fail_timeout=10s;;
        # ... 共 10 台

        # 每个 worker 保持 1000 个空闲长连接（16 workers × 1000 =
16,000）
        keepalive 1000;
    }

    server {
        listen 8080 reuseport backlog=1000000;

        location / {
            proxy_pass http://backend_pool/;
```

```
            # 启用 upstream 长连接（必须！）
            proxy_http_version 1.1;
            proxy_set_header Connection "";

            # 超时设置（根据业务心跳间隔）
            proxy_connect_timeout 15s;
            proxy_send_timeout 300s;     # ≥ keepalive_timeout
            proxy_read_timeout 300s;

            # 传递必要头
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

> ✅ *关键点说明：*
>
> - *keepalive 1000：每个 worker 维护 1000 个 upstream 空闲连接；*
> - *max_conns=1000：限制单后端最大连接数，防过载；*
> - *proxy_send/read_timeout ≥ keepalive_timeout：避免 ALB 提前关闭连接；*
> - *reuseport：实现内核级负载均衡，避免 accept 锁竞争。*

5.5.5 后端优化

- 支持长连接

5.5.6 常见问题与规避

问题	原因	解决方案
ALB 报 "too many open files"	FD 限制不足	提升 LimitNOFILE + worker_rlimit_nofile
后端连接数暴增	未启用 upstream keepalive	检查 proxy_http_version 1.1 + Connection ""
连接频繁断开	超时设置不匹配	确保 proxy_read_timeout ≥ keepalive_timeout

CPU 单核 100%	未启用 reuseport	添加 listen ... reuseport;
TLS 握手慢	未复用 session	启用 ssl_session_cache + ssl_session_tickets

5.5.7 总结

总结：**反向代理百万并发的核心公式**

百万并发 = 操作系统资源 × ALB 连接管理 × upstream 长连接复用 × 后端异步能力

层级	关键动作
OS	提升 FD、优化 TCP、关闭 swap
ALB	reuseport + 大 worker_connections + upstream keepalive
Backend	异步模型 + 长连接支持 + 状态外置
验证	压测 + 监控 + 日志分析

6 六、性能测试与问题排查

6.1 6.1 诊断命令汇总

```
# 查看 ALB 错误日志（重点关注）
tail -f ALB安装目录/logs/error.log

# 查看当前连接状态
ss -s

# 查看 ALB 进程 FD 使用
lsof -p $(pgrep alb | head -1) | wc -l

# 查看系统 FD 限制
ulimit -n
cat /proc/sys/fs/file-max

# 查看 TCP 队列溢出（关键！）
netstat -s | grep -i "listen.*overflows\|drop"

# 查看 CPU 核心负载
top

# 查看 ALB 版本
ALB安装目录/sbin/alb -V
```

6.2 6.2 错误现象：压测时大量请求返回 502 Bad Gateway

6.2.1 🔍 错误分析

ALB 无法与 upstream 后端建立连接或通信失败；

常见原因：

- 后端服务宕机或端口未监听；
- proxy_connect_timeout 过短，后端启动慢；

- 后端连接数达到上限（如 Tomcat maxConnections 耗尽）；
- DNS 解析失败（若 upstream 使用域名）。

6.2.2 ⚙️ 解决方案

```
location / {
    proxy_pass http://backend;
    proxy_connect_timeout 15s;    # 根据后端启动时间调整
    proxy_read_timeout 30s;
    resolver 8.8.8.8 valid=30s; # 若用域名，显式指定 DNS
}
```

6.2.3 ✅ 验证方法

- 查看 ALB error log：

```
grep "502" logs/error.log | head -5
# 典型日志：connect() failed (111: Connection refused)
```

- 手动 curl 测试 upstream 是否可达；
- 压测时监控后端 accept 队列：ss -lnt 查看 Recv-Q 是否溢出。

6.3 6.3 错误现象：压测初期正常，随后 QPS 骤降，连接堆积

6.3.1 🔍 错误分析

- ALB worker_connections 或系统文件描述符（FD）耗尽；
- 日志中出现 accept() failed (24: `Too many open files`)；
- 原因：默认 worker_connections=1024 + 系统 ulimit -n=1024 无法支撑高并发。

6.3.2 ⚙️ 解决方案

- 操作系统层面：

```
# 编辑alb启动脚本，脚本开头添加以下两行内容
# 【ALB安装目录】/bin/start-alb.sh或start-alb-and-console.sh
ulimit -n 100000
ulimit -Hn 100000
```

```
# 若使用 systemd，则在[Service]结尾添加LimitNOFILE=100000
# 编辑 /etc/systemd/system/alb.service
[Service]
....
User=root
Type=forking
LimitNOFILE=100000   # 添加这一行内容
# 修改完后，使用systemctl daemon-reload 然后重启ALB
```

ALB配置

```
worker_rlimit_nofile 100000;
events {
    worker_connections 65536;
}
```

6.3.3 ✅ 验证方法

- 检查进程 FD 限制：

```
cat /proc/$(pgrep alb | head -1)/limits | grep "open files"
```

- 实时监控 FD 使用量：

```
lsof -p $(pgrep alb | head -1) | wc -l
```

- 压测时观察 error log 是否仍有 “Too many open files”。

6.4 6.4. 错误现象：压测时出现大量 504 Gateway Timeout

6.4.1 🔍 错误分析

- ALB 等待 upstream 响应超时；
- 默认 proxy_read_timeout=60s，但后端处理慢或 hang 死；
- 在高并发下，少量慢请求即可拖垮整个 worker。

6.4.2 ⚙ 解决方案

- 合理设置超时（根据业务 P99 响应时间）：

```
proxy_connect_timeout 15s;
proxy_send_timeout 60s;
proxy_read_timeout 60s;
```

- 后端优化：引入异步处理、熔断机制；
- 避免长尾请求：通过限流（limit_req）保护后端。

6.4.3 ✅ 验证方法

- 故意让后端 sleep 10s，确认 ALB 在 5s 后返回 504；
- 查看error log：

```
grep "upstream timed out" error.log
```

6.5 6.5 错误现象：压测时客户端报 “Connection reset by peer” 或 “Connection closed”

6.5.1 🔍 错误分析

- ALB主动关闭了连接，常见于：
  - 客户端发送了不完整请求（如只发 header 不发 body）；
  - client_body_timeout 或 client_header_timeout 触发；
  - 后端返回非法响应（如 chunked 编码错误）。

6.5.2 ⚙️ 解决方案

- 调整客户端行为（确保完整 HTTP 请求）；
- 适当放宽超时（仅限可信内网）：

```
client_header_timeout 30s;
client_body_timeout 30s;
```

- 启用更详细的日志定位问题：

```
error_log error.log debug;
```

6.5.3 ✅ 验证方法

- 使用 tcpdump 抓包分析 RST 包来源；
- 对比开启/关闭 debug log 的错误详情。

6.6 6.6 错误现象：启用 upstream keepalive 后，QPS 无提升甚至下降

6.6.1 🔍 错误分析

- 配置遗漏：未设置 proxy_http_version 1.1 或未清除 Connection 头；
- 后端不支持 keep-alive：返回 Connection: close；
- keepalive 连接池过小：频繁新建连接，复用率低。

6.6.2 ⚙️ 解决方案

```
upstream backend {
    server 10.0.0.10:8080;
    keepalive 128;  # 根据 QPS 调整
}

location / {
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Connection "";  # 关键!
}
```

✅ 验证方法

- 检查 upstream 连接状态：
  - `ss -tnp | grep ':8080' | grep alb | grep ESTAB | wc -l`
  - 若数值稳定 ≈ worker_processes × keepalive，说明生效；
- 压测对比：开启前后 QPS 与 TIME-WAIT 数量。

6.7 6.7 错误现象：压测时系统 CPU sys%（内核态）过高，达 70%+

6.7.1 🔍 错误分析

- 频繁 syscall（如 read/write）导致上下文切换开销大；
- 未启用 sendfile、tcp_nopush 等零拷贝机制；

- 小包频繁发送（Nagle 算法未优化）。

6.7.2 ⚙ 解决方案

```
sendfile on;
tcp_nopush on;    # 累积数据后发送，减少 packet
tcp_nodelay on;   # 小包立即发（动态内容适用）
```

- 静态资源优先使用 `sendfile`；
- 动态 API 可关闭 `tcp_nopush`，启用 `tcp_nodelay`。

6.7.3 ✅ 验证方法

- 使用 `perf top` 观察内核函数（如 `tcp_sendmsg`）占比是否下降；
- 对比优化前后 `sar -u` 中 `%sys` 指标。

6.8 6.8 错误现象：多核 CPU 下，仅 CPU0 负载 100%，其他核空闲

6.8.1 🔍 错误分析

- 未启用 reuseport，所有连接由单个 worker accept；
- 网卡中断集中在 CPU0。

6.8.2 ⚙ 解决方案

```
listen 80 reuseport;
listen 443 ssl http2 reuseport;
```

6.8.3 ✅ 验证方法

```
top  # 观察各 CPU 核心负载是否均衡
ss -lnt | grep :80  # 查看是否有多个 listening socket（reuseport 特征）
```

7 七、性能测试错误速查表

错误现象	可能原因	关键检查命令	解决方案
大量 502 Bad Gateway	后端不可达 / 连接拒绝 / DNS 失败	`curl -v http://backend` `grep "connect.*failed" logs/error.log`	检查后端服务状态 设置 resolver（若用域名） 调大 `proxy_connect_timeout`
大量 `504 Gateway Timeout`	后端响应慢 / hang 死	`grep "upstream timed out" logs/error.log` `ss -tnp	grep :backend_port`	
QPS 上不去，连接堆积	FD 耗尽 / `worker_connections` 不足	`cat /proc/$(pgrep alb)/limits	grep open lsof -p $(pgrep alb)
压测报 "Too many open files"	系统或 ALB文件描述符限制过低	`dmesg	tail systemctl show alb
HTTPS QPS 远低于 HTTP	TLS 握手未复用 / cipher 低效	`openssl s_client -connect host:443 -reconnect 2>/dev/null	grep Reused`	
CPU 仅单核 100%，其余空闲	未启用 `reuseport` / `accept` 锁竞争	`top` `ss -lnt	grep :80`（看是否多个监听 socket）
启用 upstream keepalive 无效	缺少 HTTP/1.1 或 Connection 头未清除	`ss -tnp	grep :backend_port`	
客户端报 "Connection reset by peer"	超时触发 / 请求不完整 / 后端异常关闭	`tcpdump -i any -nn port 80 -w debug.pcap` `grep "client timed out" logs/error.log`	调整 `client_header_timeout` / `client_body_timeout` 确保压测工具发送完整 HTTP 请求
TIME-WAIT 连接数暴增（>10万）	未复用连接 / 主动关闭方为 ALB	`ss -s` `netstat -ant	awk '/TIME-WAIT/ {count++} END {print count+0}'`	
静态文件性能差	未启用零拷贝 / buffer 小	`strace -p $(pgrep alb) 2>&1	grep -E 'read`	

8 八、性能优化关键参数清单

表格清单说明：

- 指令： ALB的配置指令，指ALB功能配置的参数名。
- 上下文： 指该参数的作用域。

作用域标识	对应配置块	说明
main	全局配置区域	在配置文件顶层，不包含在任何块内的全局参数配置区域
events	事件模型配置块	在`events{}`块内配置，用于设置ALB事件处理模型参数
http	HTTP服务配置块	在`http{}`块内配置，用于设置HTTP服务和代理相关参数
server	虚拟服务器配置块	在`server{}`块内配置，用于设置HTTP虚拟服务器参数
location	URL匹配配置块	在`location{}`块内配置，用于设置特定URL路径的处理参数
upstream	上游服务配置块	在`upstream{}`块内配置，用于设置反向代理的后端服务器组参数
stream	四层代理配置块	在`stream{}`块内配置，用于设置TCP/UDP四层代理参数
main/http/server/location	多级配置区域	可在main、http、server、location任意层级中配置
http/server/location	HTTP配置区域	仅可在http、server、location层级中配置

8.1 8.1 全局与进程控制

指令	上下文	功能说明	默认值	建议值	注意事项
`worker_processes`	main	ALB worker 进程数	auto	auto 或 CPU 核数（如 8）	auto 自动匹配 CPU 核心数；多核必设
`worker_rlimit_nofile`	main	单个 worker 最大文件描述符数	无限制（继承系统）	65536 ~ 262144	必须 ≥ worker_connections × 2（client + upstream）
`worker_cpu_affinity`	main	绑定 worker 到指定 CPU 核	未绑定	auto（或手动掩码）	减少上下文切换，提升缓存命中率（可选）

8.2 8.2 事件模型（Events）

指令	上下文	功能说明	默认值	建议值	注意事项
worker_connections	events	单个 worker 最大并发连接数	512	10000 ~ 200000	总并发 ≈ worker_processes × worker_connections
use	events	事件驱动模型	自动选择	epoll（Linux）	Linux 下必须显式设为 epoll（虽默认但建议写明）
multi_accept	events	一次 accept 多个连接	off	on	配合 epoll 提升吞吐，避免饥饿

8.3 8.3 HTTP 全局配置

指令	上下文	功能说明	默认值	建议值	注意事项
sendfile	http/server/location	零拷贝传输静态文件	off	on	静态资源服务必开，减少内核/用户态拷贝
tcp_nopush	http/server/location	累积数据后发送（配合 sendfile）	off	on	提升大文件传输效率，减少小包
tcp_nodelay	http/server/location	禁用 Nagle 算法，小包立即发	off	on（API 场景）	动态内容/API 建议开启，降低延迟
open_file_cache	http/server/location	缓存文件元信息（fd/stat）	off	max=10000 inactive=60s	高频静态文件访问必备
open_file_cache_valid	http/server/location	缓存验证间隔	60s	30s ~ 120s	配合 open_file_cache 使用
open_file_cache_min_uses	http/server/location	文件最少访问次数	1	2	防止缓存冷文件

		才缓存			

8.4 8.4 客户端连接与超时

指令	上下文	功能说明	默认值	建议值	注意事项
`keepalive_timeout`	http/server/location	客户端 keep-alive 连接保持时间	75s	15s ~ 30s（Web API） 300s（长连接）	过大会导致 idle 连接堆积
`keepalive_requests`	http/server/location	单连接最大请求数	100	1000 ~ 1000000	防止单连接内存泄漏
`client_header_timeout`	http/server/location	读取请求头超时	60s	10s ~ 30s 高并发下可以增大如 120s	防 Slowloris 攻击
`client_body_timeout`	http/server/location	读取请求体超时	60s	10s ~ 120s 文件上传可根据实际时间设置	大文件上传需调大，API 可调小
`reset_timedout_connection`	http/server/location	超时后发送 RST 而非 FIN	off	on	快速释放 TIME-WAIT（谨慎使用）

8.5 8.5 反向代理（Proxy）

指令	上下文	功能说明	默认值	建议值	注意事项
`proxy_http_version`	http/server/location	代理使用的 HTTP 版本	1.0	1.1	启用 upstream keepalive 必须设为 1.1
`proxy_set_header Connection ""`	location	清除 Connection 头	无	`""`	配合 keepalive 使用，防止 close

`proxy_connect_timeout`	http/server/location	与 upstream 建连超时	60s	1s ~ 10s 高并发下可以增大如120s	后端健康检查快失败
`proxy_send_timeout`	http/server/location	向 upstream 发送请求超时	60s	5s ~ 30s 高并发下可以增大如120s	根据业务 P99 设置
`proxy_read_timeout`	http/server/location	从 upstream 读响应超时	60s	5s ~ 300s 高并发或耗时业务可以增大	长连接场景可设大
`proxy_buffering`	http/server/location	是否缓冲后端响应	on	on（默认）	关闭会增加后端压力，一般不关
`proxy_buffer_size`	http/server/location	第一块 buffer 大小	4k|8k	16k	包含响应头，建议 ≥16k
`proxy_buffers`	http/server/location	响应体 buffer 数量与大小	8 4k|8k	8 32k	高吞吐场景适当增大

8.6 8.6 Upstream 长连接池

指令	上下文	功能说明	默认值	建议设置范围	注意事项
`keepalive`	upstream	每个 worker 的空闲长连接数	无	32 ~ 256	总连接池 = `worker_processes` × `keepalive`
`keepalive_requests`	upstream	单个 upstream 连接最大请求数	1000	1000（默认，合理）	防止连接内存泄漏
`keepalive_timeout`	upstream	upstream 空闲连接超时	60s	60s（默认，合理）	通常无需修改

8.7 8.7 日志

指令	上下文	功能说明	默认值	建议值	注意事项
`access_log`	http/server/location	访问日志	`logs/access.log combined`	`off`（压测）或 `buffer=64k flush=5s`	高并发下关闭或缓冲写入
`error_log`	main/http/server	错误日志级别	`error`	`warn`（生产）`crit`（压测）	避免 debug 日志拖慢性能

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com