云计算高可用架构实战记录

2025 年 7 月 25 日

摘要

本文档系统性记录了基于 CentOS Stream 和 MariaDB 构建 WordPress 高可用（High Availability, HA）架构的各阶段实践过程，涵盖从 L1 到 L4 的逐步演进，旨在总结经验，并同时为日后实践操作作参考文档。

目录

1 项目实施阶段概述

本次挑战任务按照以下四个阶段逐级推进：

- **L1 基础架构部署（Single Web + Single DB + 单节点共享存储）**
- **L2 Web 层高可用：双 Web 节点 + 负载均衡（Load Balancing）**
- **L3 数据库高可用：主从同步（Master-Slave Replication）+ 故障切换（Failover）**
- **L4 存储高可用：双存储节点 + 数据自愈机制（Self-Healing）**

2 各阶段实现过程

2.1 L1 基础架构部署

2.1.1 概述

目标：实现基础 WordPress 网站部署、数据库连接与文件共享测试。
实施要点：

- 环境准备：三台虚拟服务器分别安装 Centos Stream9 系统
- IP 地址规划：Web 服务器 IP 为 59.110.162.172，数据库服务器 IP 为 47.94.105.235，存储服务器 IP 为 1.94.131.251。
- 组件：NGINX + PHP + WordPress，MariaDB，NFS 服务端。

2.1.2 部署流程

Web 服务器部署：

1. 系统初始化和 Web 相关组件配置：

   设置主机名、更新系统、安装必要软件包并配置防火墙：

命令 *1*: 初始化系统

```
sudo hostnamectl set-hostname web01
sudo dnf update -y
sudo dnf install -y vim wget net-tools firewalld
sudo systemctl enable --now firewalld
sudo firewall-cmd --add-service=http --permanent
sudo firewall-cmd --add-service=https --permanent
sudo firewall-cmd --reload
```

   配置 NGINX 和 PHP：

命令 *2*: 安装 NGINX 和 PHP

```
sudo dnf install -y nginx
sudo systemctl enable --now nginx
sudo dnf install -y php php-fpm php-mysqlnd php-gd php-json
   php-mbstring php-xml php-zip php-curl php-opcache
sudo systemctl enable --now php-fpm
```

配置 NGINX 配置文件：

命令 *3*: NGINX 配置

```
sudo vim /etc/nginx/conf.d/wordpress.conf
server {
    listen 80;
    server_name _; # IP访问，暂不指定域名

    root /var/www/html/wordpress;
    index index.php index.html;

    location / {
    try_files $uri $uri/ /index.php?$args;
 }

location ~ \.php$ {
fastcgi_pass unix:/run/php-fpm/www.sock;
fastcgi_param SCRIPT_FILENAME
   $document_root$fastcgi_script_name;
fastcgi_index index.php;
include fastcgi_params;
}
}
sudo systemctl restart nginx php-fpm
```

安装配置 WordPress：

命令 *4*: 安装 WordPress

```
sudo mkdir -p /var/www/html
cd /var/www/html
sudo wget https://cn.wordpress.org/latest-zh_CN.tar.gz
sudo tar -xzf latest-zh_CN.tar.gz
sudo mv wordpress/* /var/www/html/wordpress/
```

```
6       sudo rm -rf latest-zh_CN.tar.gz
7       sudo chown -R nginx:nginx /var/www/html/wordpress
8       sudo chmod -R 755 /var/www/html/wordpress
```

图 1: WordPress 安装界面

2. 配置 MariaDB 数据库

系统初始化和 MariaDB 组件安装：

命令 *5*: MariaDB 安装

```
1       sudo hostnamectl set-hostname db01
2       sudo dnf update -y
3       sudo dnf install -y vim wget net-tools firewalld
4       sudo systemctl enable --now firewalld
5       sudo firewall-cmd --add-service=mysql --permanent
6       sudo firewall-cmd --reload
7       sudo dnf install -y mariadb-server
8       sudo systemctl enable --now mariadb
```

数据库初始化和配置：

命令 *6*: MariaDB 初始化

```
1       sudo mysql_secure_installation
```

```
2       Enter current password for root (enter for none): #设置Root密
            码
3       Switch to unix_socket authentication [Y/n] #选择n, 设置为Yes后
            数据库只能通过sudo mysql连接
4       Remove anonymous users? [Y/n] Y #删除匿名用户
5       Disallow root login remotely? [Y/n] N #允许远程Root登录, 生产
            环境建议禁用
6       Remove test database and access to it? [Y/n] Y #删除测试数据库
7       Reload privilege tables now? [Y/n] Y #重新加载权限表
```

创建 WordPress 数据库和用户：

命令 7: 创建 WordPress 数据库

```
1       sudo mysql -u root -p
2       # 在shell中执行以下:
3       CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8mb4
            COLLATE utf8mb4_general_ci;
4       CREATE USER 'wpuser'@'59.110.162.172' IDENTIFIED BY '
            StrongPassword123';
5       GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@
            '59.110.162.172';
6       FLUSH PRIVILEGES; # 刷新权限
7       EXIT;
```

MySQL 语法：

- CREATE DATABASE: 创建新数据库
- CREATE USER 'user''host' IDENTIFIED BY 'pwd': 创建用户
- GRANT 权限 ON 数据库. 表 TO 用户: 授予用户权限
- REVOKE 权限 FROM 用户: 撤销权限
- DROP DATABASE dbname: 删除数据库
- DROP USER 'user''host': 删除用户
- SHOW DATABASES;: 列出所有数据库
- USE dbname;: 切换当前数据库
- SHOW TABLES;: 列出当前数据库中的表
- DESCRIBE tablename;: 查看表结构

配置远程访问：

命令 *8*: 配置远程访问

```
sudo vim /etc/my.cnf.d/mariadb-server.cnf
```

取消以下部分的注释：

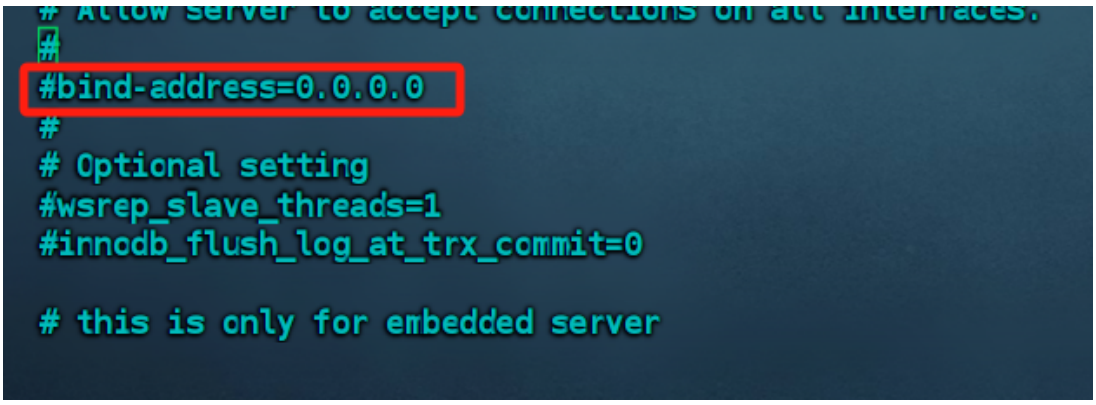

图 2: MariaDB 配置文件修改

使用`sudo systemctl restart mariadb`, 重启 MariaDB 服务。

在 Web 节点执行以下指令测试连接：

命令 *9*: 测试数据库连接

```
sudo dnf install -y mariadb
mysql -h 47.94.105.235 -u wpuser -p
# 输入密码 StrongPassword123
```

出现以下图表示连接成功：

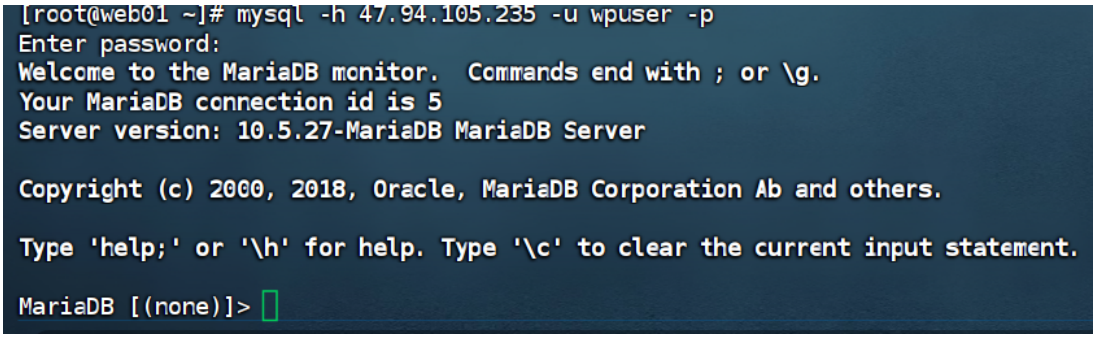

图 3: MariaDB 连接测试成功

输入以下信息完成 WordPress 安装向导：

图 4: WordPress 安装向导

如果提示“无法创建配置文件”，请手动创建 wp-config.php 文件：

命令 *10*: 手动创建 wp-config.php

```
cd /var/www/html/wordpress
sudo vim wp-config.php
#粘贴网站提供的配置内容
sudo chown nginx:nginx wp-config.php
sudo chmod 640 wp-config.php
```

接着完成信息填写，安装 WordPress 程序

3. 配置 NFS 共享存储：初始化系统和 NFS 组件安装：

命令 *11*: NFS 安装

```
sudo hostnamectl set-hostname nfs01
sudo dnf update -y
sudo dnf install -y nfs-utils vim wget net-tools firewalld
sudo systemctl enable --now firewalld
sudo firewall-cmd --add-service=nfs --permanent
sudo firewall-cmd --reload
```

创建共享目录并配置权限：

命令 *12*: 配置 NFS 共享目录

```
sudo mkdir -p /mnt/wp-shared
sudo chown -R nobody:nobody /mnt/wp-shared
sudo chmod -R 777 /mnt/wp-shared
# 777权限允许所有用户读写执行.
# nobody:nobody 是NFS默认用户组
sudo vim /etc/exports
/mnt/wp-shared 59.110.162.172(rw,sync,no_root_squash,
   no_subtree_check)
sudo exportfs -arv # 重新导出共享目录
sudo exportfs -v # 查看导出目录
```

关于 NFS 共享参数的简表：参考表1。

在 Web01 节点挂载 NFS 共享目录：

命令 *13*: 挂载 NFS 共享目录

```
sudo dnf install -y nfs-utils
sudo mkdir -p /var/www/html/wordpress/wp-content/uploads
sudo mount -t nfs 1.94.131.251:/mnt/wp-shared /var/www/html/
   wordpress/wp-content/uploads
# 挂载后检查目录
df -h | grep uploads
```

4. 测试部署：

- 访问 WordPress 网站：在浏览器中输入 http://59.110.162.172/ 出现 WordPress 主页内容，说明 Web 服务正常。
- 登录 WordPress 后台：新建一篇文章，退出后台、重新登录，检查文章是否保存成功。保存成功则说明数据库连接正常。
- 上传媒体文件：在 WordPress 后台尝试上传图片，检查相应文件是否成功存储在 NFS 共享目录（/mnt/wp-shared）中。存在证明 NFS 挂载成功。

经过测试，L1 基础架构部署成功，WordPress 网站可以正常访问，数据库连接正常，NFS 共享存储功能可用。

2.1.3 问题解决记录

WordPress 媒体上传失败问题分析与修复

在 L1 基础架构部署过程中，WordPress 系统基本功能正常，但在后台尝试上传图片时遇到以下错误：

无法创建目录 *wp-content/uploads/2025/07*。它的父目录是否可以被服务器写入？

问题分析：

该错误发生在尚未配置 NFS 共享存储阶段，理论上应能正常写入 Web 节点本地目录，但实际操作失败。

排查过程：

1. **权限检查：**

命令 *14*: 检查目录权限

```
ls -ld /var/www/html/wordpress/wp-content/uploads
# 输出: drwxr-xr-x nginx nginx ...
```

目录权限和属主设置正确，初步排除权限问题。

2. **进程用户检查：**

命令 *15*: 检查 PHP-FPM 运行用户

```
ps aux | grep php-fpm | head -3
# 发现: apache用户运行，而非nginx
```

3. **根因确定：**PHP-FPM 以 `apache` 用户运行，而 `uploads` 目录属主为 `nginx`，导致写入权限不足。

解决方案：

统一 PHP-FPM 与 NGINX 运行用户：

命令 *16*: 修改 PHP-FPM 配置

```
sudo vim /etc/php-fpm.d/www.conf
# 修改以下行:
user = nginx
group = nginx

# 重启服务
sudo systemctl restart php-fpm

# 验证修改
ps aux | grep php-fpm | head -3
```

验证结果：

修复后，WordPress 媒体上传功能恢复正常，成功创建 `wp-content/uploads/2025/07/` 目录并完成文件上传。

经验总结：

此问题凸显了 Web 服务架构中用户权限一致性的重要性。建议在初始部署时：

- 统一 Web Server 与 PHP-FPM 的运行用户
- 部署完成后立即进行文件上传功能测试
- 避免混合使用不同用户权限，防止后续权限冲突

2.1.4 总结与反思

以下是 L1 阶段的主要收获与经验：

选项	默认	说明
rw / ro	ro	是否允许读写（rw 可写，ro 只读）
sync / async	async	是否同步写入（sync 更安全，async 性能高但有丢数据风险）
root_squash / no_root_squash	root_squash	是否将客户端 root 权限映射为 nobody 用户
subtree_check / no_subtree_check	subtree_check	是否检查子目录结构（建议关闭提升性能）

表 1: NFS 共享参数简表

了解 Web/DB/存储三层职责分离、完成了 LNMP 环境部署、WordPress 安装与配置、MariaDB 数据库创建与用户授权、NFS 共享存储配置等关键技能。

上面的 StongPassword123 是一个示例密码，实际部署要切换密码。数据库可以修改 0.0.0.0 为 Web 节点的 IP 59.110.162.172 ，提升安全性。

对于 NFS 共享存储，可以修改 nobody:nobody 为 nginx:nginx，777 权限可以按需修改，更准确文件控制权限。Web 节点上的挂载持久化可以按照以下步骤进行：

命令 *17*: NFS 持久化挂载

```
sudo vim /etc/fstab
# 添加以下一行
1.94.131.251:/mnt/wp-shared /var/www/html/wordpress/wp-content/uploads
    nfs defaults,_netdev 0 0
sudo umount /var/www/html/wordpress/wp-content/uploads # 卸载当前挂载
sudo mount -a # 重新挂载所有fstab中的文件系统
df -h | grep uploads # 验证挂载成功
```

之后重启系统，检查挂载是否依旧有效。

2.2 L2 Web 层高可用实施

2.2.1 概述

目标：实现前端服务不中断，支持 Web 节点故障切换。

方案要点：

- 增设第二台 Web 节点，实现双机热备或负载均衡架构。
- 引入 HAProxy 或 Keepalived 实现虚拟 IP（VIP）管理与流量调度。
- 共享存储目录配置一致性保障。

IP 规划：新增 Web 节点 IP 为 154.9.253.137，新增 VIP 节点为 155.133.7.217.
架构图：

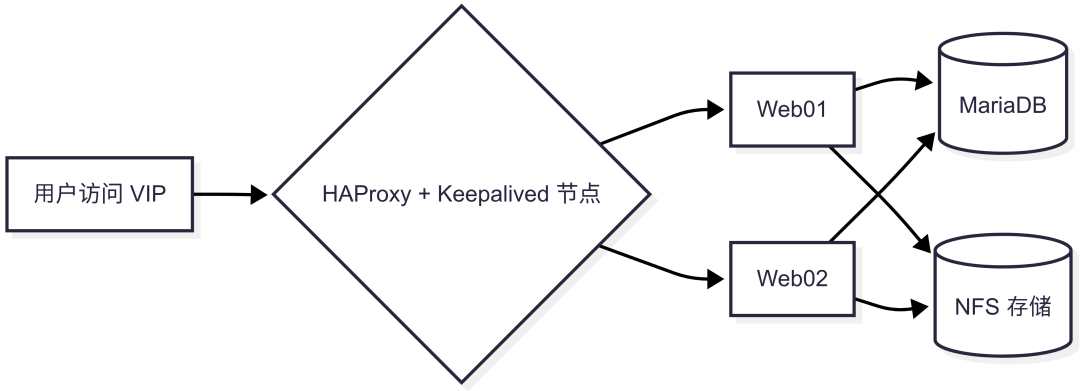

图 5: L2 Web 层高可用架构图

2.2.2 实操步骤

1. 设置主机名、安装相关软件包、配置防火墙和 SELinux：

命令 *18*: Web 节点初始化

```
# 设置主机名
hostnamectl set-hostname web02
# 更新系统
dnf update -y
# 安装基础工具
dnf install -y vim net-tools wget curl bash-completion lsof
# 配置时间同步
dnf install -y chrony
systemctl enable --now chronyd
# 关闭防火墙和 SELinux（实验环境下推荐，Not in 生产模式）
systemctl stop firewalld && systemctl disable firewalld
setenforce 0
sed -i 's/^SELINUX=.*/SELINUX=disabled/' /etc/selinux/config
```

2. 安装 NGINX 和 PHP-FPM：

命令 *19*: 安装 NGINX 和 PHP-FPM

```
dnf install -y epel-release
dnf module reset php -y
dnf module enable php:8.1 -y
dnf install -y php php-mysqlnd php-fpm php-gd php-mbstring
   php-xml php-json php-opcache php-curl
dnf install -y nginx
systemctl enable --now nginx
sed -i 's/^user = apache/user = nginx/' /etc/php-fpm.d/www.
   conf
sed -i 's/^group = apache/group = nginx/' /etc/php-fpm.d/www.
   conf
systemctl enable --now php-fpm
```

其中，dnf module 指令是 CentOS 8 及以上版本的模块化软件包管理功能，允许用户选择特定版本的软件包。

3. 配置 Wordpress 数据库和 NFS 共享存储：

命令 *20*: 配置数据库和 NFS

```
#编辑wp-config.php保障与Web01一致
vim /var/www/html/wordpress/wp-config.php
define( 'DB_NAME', 'wordpress' );
define( 'DB_USER', 'wpuser' );
define( 'DB_PASSWORD', 'StrongPassword123' );
define( 'DB_HOST', '47.94.105.235' ); // MariaDB 节点地址
dnf install -y nfs-utils

#首先在nfs01的/etc/exports中添加以下行
/mnt/wp-shared 154.9.253.137(rw,sync,no_root_squash,
   no_subtree_check)
#然后在Web02节点执行以下命令挂载NFS共享目录
mkdir -p /var/www/wordpress/wp-content/uploads
mount -t nfs 1.94.131.251:/mnt/wp-shared /var/www/wordpress/
   wp-content/uploads
chown -R nginx:nginx /var/www/wordpress/wp-content/uploads
sudo vim /etc/fstab
# 添加以下一行，持久化挂载
1.94.131.251:/mnt/wp-shared /var/www/wordpress/wp-content/
   uploads nfs defaults,_netdev 0 0
```

注：此处省略了 WordPress 安装步骤，配置 NGINX 文件见 L1。

4. VIP 配置与 HAProxy 安装

初始化系统：

命令 *21*: VIP 配置与 HAProxy 安装

```
# 设置主机名
hostnamectl set-hostname lb01

# 系统更新 & 工具
dnf update -y
dnf install -y vim wget curl net-tools lsof bash-completion

# 时间同步
dnf install -y chrony
systemctl enable --now chronyd

# 设置 SELinux 为关闭
setenforce 0
sed -i 's/^SELINUX=.*/SELINUX=disabled/' /etc/selinux/config

#配置防火墙
dnf -y install firewalld
systemctl enable --now firewalld
firewall-cmd --add-port=80/tcp --permanent
firewall-cmd --permanent --add-rich-rule='rule protocol value="
   vrrp" accept'
firewall-cmd --reload
```

安装 HAProxy：

命令 *22*: 安装 HAProxy

```
    dnf install -y haproxy
    systemctl enable --now haproxy
```

配置 HAProxy：

命令 *23*: HAProxy 配置

```
# 编辑配置文件
cat >/etc/haproxy/haproxy.cfg <<'EOF'
global
```

```
    log /dev/log local0
    maxconn 2048
    daemon

defaults
    log    global
    option httplog
    option dontlognull
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    maxconn 1024

frontend http_front
    bind *:80
    mode http
    default_backend web_servers

backend web_servers
    balance roundrobin
    mode http
    option httpchk GET /
    server web01 59.110.162.172:80 check
    server web02 154.9.253.137:80 check
EOF

# 检查配置并启动，上述负载均衡为轮询
haproxy -c -f /etc/haproxy/haproxy.cfg
systemctl enable --now haproxy
```

5. 安装 Keepalived 并配置 VIP

命令 *24*: 安装 Keepalived

```
        dnf install -y keepalived
        # 编辑 Keepalived 配置文件
        vim /etc/keepalived/keepalived.conf
        # 添加以下内容
        vrrp_instance VI_1 {
        state MASTER
```

```
        interface eth0          #  替换为当前网卡名
        virtual_router_id 51
        priority 100
        advert_int 1
        authentication {
        auth_type PASS
        auth_pass 1234
        }
        virtual_ipaddress {
        155.133.7.217
        }
    }

        systemctl enable --now keepalived
```

6. 配置站点 URL 并测试 VIP 访问

命令 *25*: 配置站点 URL

```
        # 编辑 wp-config.php
        vim /var/www/html/wordpress/wp-config.php
        //VIP（没有域名，就修改成155.133.7.217）
        define('WP_HOME', 'https://test.jiepress.me');
        define('WP_SITEURL', 'https://test.jiepress.me');
        # 测试访问
        curl -I https://test.jiepress.me
        # 应返回 HTTP/1.1 200 OK
```

为每台 web 添加 health check：

命令 *26*: 添加健康检查

```
        # 两台web执行
        cat >/var/www/wordpress/health.php <<EOF #wordpress 根目录
        <?php
        http_response_code(200);
        echo "OK";
        EOF
        chown nginx:nginx /var/www/wordpress/health.php
```

在下图二处修改 haproxy 配置文件：

7. 测试负载均衡与故障切换

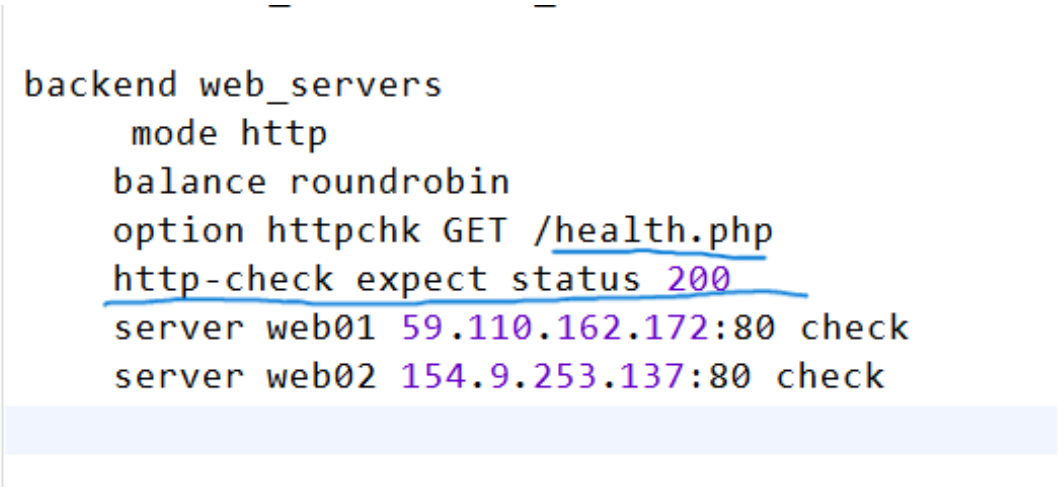

图 6: HAProxy 健康检查配置

- 访问 VIP 地址，检查是否能正常加载 WordPress 页面。
- 在 Web01 上停止 NGINX 服务，验证 HAProxy 是否能自动切换到 Web02。

经过测试，L2 阶段的 Web 层高可用架构成功实现，Web01 和 Web02 节点可以通过 VIP，故障切换正常工作。

2.2.3 问题解决记录

NGINX 配置文件问题：

在已经设置了 NGINX 配置文件（/etc/nginx/conf.d/wordpress.conf）情况下，发现访问 VIP 时依旧只出现 CentOS 的默认配置页。

问题分析：经过检查发现，NGINX 默认配置文件（/etc/nginx/nginx.conf）配置文件中 server 设置为：default_server，导致访问 VIP 时优先加载默认配置。见图7。

解决方案：修改 NGINX 配置文件，将 server 中的 default_server 删除，确保配置文件正确加载。

MariaDB 连接失败：

在 Web 节点配置 WordPress 并测试数据库连接时，出现连接错误：[ERROR 1045] Access denied for user 'wpuser'@'x.x.x.x' (using password: YES)。

问题分析：检查防火墙设置、数据库服务状态均正常。进一步排查发现，MariaDB 用户 `wpuser` 仅授权给特定主机（如 web1），未包含当前 Web 节点的 IP 地址，导致远程连接失败。

解决方案：登录数据库服务器，重新为 `wpuser` 授权指定 Web 节点的 IP 地址（如 59.110.162.172）：

命令 *27*: 授权 wpuser

```
1  GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@'59.110.162.172'
       IDENTIFIED BY 'StrongPassword123';
2  FLUSH PRIVILEGES;
```

修改后重新连接，问题解决。

```
server {
    listen       80 default_server;
    listen       [::]:80 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
        location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
        location = /50x.html {
    }
}
```

图 7: NGINX 默认配置文件示例

手机访问 VIP 失败（unreachable）：

在使用 HAProxy 实现 Web 层高可用架构后，PC 端访问 VIP 的公网 IP 一切正常，但手机端（包括 Wi-Fi 和流量）访问同一 IP 地址时提示 `unreachable` 或 `ERR_CONNECTION_REFUSED`，初步以为是网络、防火墙或服务未启动问题。

问题分析：

经多轮排查，确认如下：

- 云服务器 HAProxy 服务正常，监听端口为 `*:80`；
- 防火墙及云平台安全组均已放行 `TCP:80` 端口；
- 手机端使用 `curl` 命令访问 VIP 公网 IP 正常返回 HTML 内容；
- 仅在使用浏览器（如 Chrome）访问 `http://155.133.7.217` 时出现失败。

进一步测试发现：

- Firefox 浏览器访问该 IP 正常；
- Chrome 浏览器对 `HTTP + 纯 IP 地址`的访问存在安全限制，可能强制升级为 HTTPS 或直接拦截。

该问题并非架构配置错误，而是移动端浏览器的安全策略行为导致。

解决方案：

为 HAProxy 所在云主机绑定一个合法的二级域名（如通过 Cloudflare 设置 A 记录），并通过该域名访问 VIP，示例：

```
https://test.jiepress.me
```

此后，手机端浏览器访问恢复正常，问题解决。

2.2.4 总结与反思

L2 阶段引入了第二台 Web 节点并构建基于 HAProxy 和 Keepalived 的流量调度机制，成功实现了 Web 层的高可用架构。以下是主要收获与经验：

1. 不要忘记对 MySql 数据库用户授权，确保 Web 节点可以正常连接数据库。每一个 Web 节点都是一个单独的实体。
2. 检查 Nginx 默认配置文件，确保正确加载自定义站点配置。
3. HAProxy 的健康检查配置可以有效监控 Web 节点状态，实现自动故障切换。
4. 手机端浏览器访问 IP 地址时可能存在安全限制，建议使用域名访问。

2.3 L3 数据库高可用实施

2.3.1 概述

目标：构建主从数据库复制机制，并实现主库故障后的自动接管。

关键技术：

- MariaDB 主从复制（Replication）配置。
- 使用 MaxScale 自动检测主库状态并进行故障切换。
- 数据一致性验证：主库插入的数据，在切换后从库可读。

架构图：

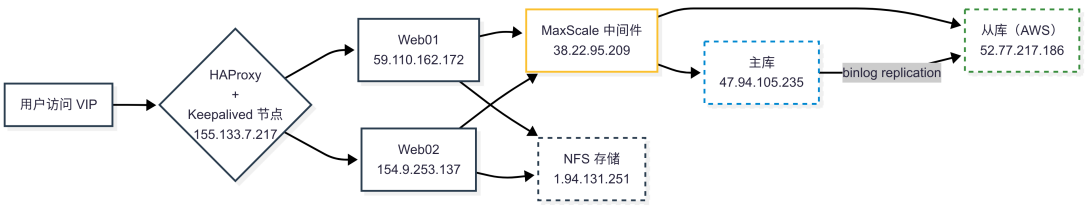

图 8: L3 数据库高可用架构图

IP 规划：新增数据库从节点 IP 为 52.77.217.186 ，MaxScale 节点 IP 为 47.94.105.235。

2.3.2 实操步骤

实施方案：

1. 主库配置

修改配置文件 /etc/my.cnf.d/mariadb-server.cnf，如下图：

命令 *28*: 主库配置

```
[mysqld]
server-id=1
log-bin=mysql-bin
binlog-format=mixed
gtid_strict_mode=ON
```

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
log-error=/var/log/mariadb/mariadb.log
pid-file=/run/mariadb/mariadb.pid

# 主库配置（添加）
server-id=1
log-bin=mysql-bin
binlog-format=mixed
gtid_strict_mode=ON
```

图 9: MariaDB 主库配置

创建复制和探测用户：

命令 *29*: 创建复制用户

```
-- 复制账户（供从库使用）
CREATE USER 'repl'@'%' IDENTIFIED BY 'replpass';
GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%';

-- MaxScale 探测账户
CREATE USER 'maxuser'@'38.22.95.209' IDENTIFIED BY 'maxpass';
GRANT REPLICATION CLIENT, REPLICATION SLAVE, SLAVE MONITOR,
   READ_ONLY ADMIN ON *.* TO 'maxuser'@'38.22.95.209';
GRANT SHOW DATABASES ON *.* TO 'wpuser'@'38.22.95.209';
GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@
   '38.22.95.209';
```

```
        FLUSH PRIVILEGES;
        # 查看当前状态
        SHOW VARIABLES LIKE 'log_bin';
        SHOW MASTER STATUS;
```

重启主库服务。

2. 从库配置

在从库节点（52.77.217.186）上安装 MariaDB 并配置：

命令 *30*: 从库配置

```
        hostnamectl set-hostname db02
        dnf update -y
        dnf install -y mariadb-server vim wget net-tools firewalld
        systemctl enable --now firewalld
        firewall-cmd --add-service=mysql --permanent
        firewall-cmd --reload
        systemctl enable --now mariadb
        vim /etc/my.cnf.d/mariadb-server.cnf
        server-id=2
        log-bin=mysql-bin
        binlog-format=mixed
        gtid_strict_mode=ON
        systemctl restart mariadb
```

执行主从复制连接：

命令 *31*: 主从复制连接

```
        STOP SLAVE;

        CHANGE MASTER TO
        MASTER_HOST='47.94.105.235',
        MASTER_USER='repl', # 与之前一致
        MASTER_PASSWORD='replpass', # 与之前一致
        MASTER_LOG_FILE='mysql-bin.000001', # 主库日志文件名
        MASTER_LOG_POS=328, # 主库日志位置
        MASTER_USE_GTID = slave_pos;

        START SLAVE;
        SHOW SLAVE STATUS\G
```

```
        # 确认 Slave_IO_Running 和 Slave_SQL_Running 均为 Yes

        # 如果失败，大概率提示
        Error executing row event: 'Table 'wordpress.wp_options'
           doesn't exist'
        # 在主库执行以下dump导出sql文件
        mysqldump -uroot -p --all-databases --master-data=2 --single-
           transaction --flush-logs --hex-blob > full.sql
        # 将 full.sql 传输到从库，在从库执行
        mysql -uroot -p < full.sql
```

3. MaxScale 安装与配置

安装 MaxScale

命令 *32*: 安装 MaxScale

```
        hostnamectl set-hostname maxscale01
        curl -sSL https://downloads.mariadb.com/MariaDB/
           mariadb_repo_setup | sudo bash
        sudo dnf install maxscale -y

        #防火墙配置
        sudo firewall-cmd --permanent --add-port=3306/tcp
        sudo firewall-cmd --reload

        sudo systemctl enable --now maxscale
        ss -tlnp | grep 3306
```

配置 MaxScale 文件

命令 *33*: MaxScale 配置

```
        sudo vim /etc/maxscale.cnf
        # ------------------------------
        # MaxScale 配置文件
        #  运行节点：38.22.95.209
        # ------------------------------

        [maxscale]
        threads=auto

        # 监控模块
```

```
[replication_monitor]
type=monitor
module=mariadbmon
servers=db1,db2
user=maxuser
password=maxpass
monitor_interval=2000ms
auto_failover=true
auto_rejoin=true
enforce_read_only_slaves=true


# 读写分离服务
[Read-Write-Service]
type=service
router=readwritesplit
servers=db1,db2
user=wpuser
password=StrongPassword123
max_slave_connections=1

# 监听入口（端口3306）
[Read-Write-Listener]
type=listener
service=Read-Write-Service
protocol=MariaDBClient
port=3306

# 主库
[db1]
type=server
address=47.94.105.235
port=3306
protocol=MariaDBBackend

# 从库
[db2]
type=server
address=52.77.217.186
```

```
50          port=3306
51          protocol=MariaDBBackend
```

启动 MaxScale 服务并检查状态：

命令 *34*: 启动 MaxScale

```
1           sudo systemctl restart maxscale
2           sudo systemctl status maxscale
3           # 检查 MaxScale 状态
4           maxctrl list servers
```

接着分别在 Web01 和 Web02 节点上修改 wp-config.php，指向 MaxScale 节点 (38.22.95.209)
另：正常情况见下图：

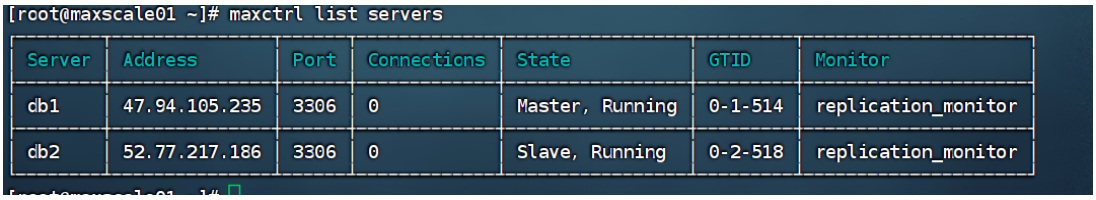

图 10: MaxScale 正常状态

4. 测试数据库高可用

在主库停止 MySQL 服务，模拟主库故障。之后在 MaxScale 节点执行以下命令，检查从库是否接管：

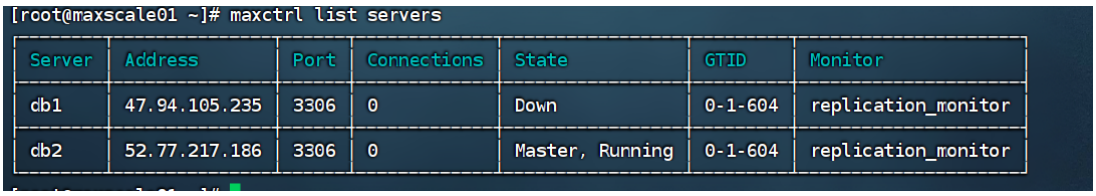

图 11: MaxScale 故障切换示例

可以看到，MaxScale 自动将流量切换到从库（db2），此时网站访问正常。

有时候，在主节点执行指令成功，但备节点失败。会导致 GTID 不一致，导致从库无法启动。这种情况下，可以在从库执行以下指令：

命令 *35*: 从库 GTID 同步

```
1           STOP SLAVE;
2           SET GLOBAL sql_slave_skip_counter = 1; # 此处的1应该根据其中一
                个大的比小的大多少来设置
3           START SLAVE;
```

```
4         SHOW SLAVE STATUS\G
5         # 确认 Slave_IO_Running 和 Slave_SQL_Running 均为 Yes
```

2.3.3 排错记录

错误一：监控用户（maxuser）的精确授权

MaxScale 的监控模块（mariadbmon）需要特定权限来检查后端数据库的主从状态、GTID、只读状态等。仅仅授予基础的连接权限是不够的。

- **问题发现：** `maxctrl list servers` 显示 `Auth Error`。后续日志显示，执行 `SHOW SLAVE STATUS` 和设置 `read_only` 状态时权限不足。
- **核心需求：**监控用户不仅需要能连接，还需要执行状态检查和配置管理的权限。
- **最终解决方案：**必须为 `maxuser` 授予以下一组权限。其中 `SLAVE MONITOR` 和 `READ_ONLY ADMIN` 是通过分析日志发现的关键权限。

命令 *36*: 授权 maxuser

```
1  -- 授予 `maxuser` 从 MaxScale 服务器登录的权限
2  -- 并赋予其监控复制状态和管理只读状态所需的所有权限
3  GRANT REPLICATION CLIENT, REPLICATION SLAVE, SLAVE MONITOR, READ_ONLY
       ADMIN
4  ON *.* TO 'maxuser'@'38.22.95.209';
5  FLUSH PRIVILEGES;
```

错误二：应用用户（wpuser）与来源主机的验证

数据库的权限模型是基于 **'用户名 @' 来源主机''** 的元组。一个用户可以从某个 IP 地址成功登录，不代表可以从另一个 IP 地址登录。

- **问题发现：**应用直接连接数据库时正常，但 MaxScale 连接后端数据库时日志显示 `Access denied for user 'wpuser'@'38.22.95.209'`。
- **核心概念：**必须为应用用户授权，允许其从 **MaxScale 服务器的 IP 地址**发起连接。
- **最终解决方案：**在所有后端数据库上，为 `wpuser` 创建一个针对 MaxScale 主机来源的授权记录。

命令 *37*: 授权 wpuser

```
1  -- 授权 `wpuser` 可以从 MaxScale 服务器登录，并操作 `wordpress` 数据库
2  GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@'38.22.95.209'
3  IDENTIFIED BY 'StrongPassword123';
```

错误三：MaxScale 内部认证机制的权限需求

为了高效地认证前端应用的连接请求，MaxScale 需要读取后端的系统权限表（如 `mysql.user`, `mysql.db`）。

- **问题发现：** WordPress 连接 MaxScale 时提示“建立数据库连接时出错”。日志显示 `SELECT command denied to user 'wpuser'@'...' for table `mysql`.`user`.`
- **核心需求：** MaxScale 的服务用户（在此案例中是 `wpuser`）需要对 `mysql` 系统数据库有只读权限。
- **最终解决方案：** 为 `wpuser` 授予对 `mysql` 数据库的 `SELECT` 权限。

命令 *38*: 授权 wpuser 读取 mysql 系统表

```
-- 授予 `wpuser` 读取 mysql 系统表的权限，以供 MaxScale 进行认证
GRANT SELECT ON mysql.* TO 'wpuser'@'38.22.95.209';
```

2.3.4 总结与反思

L3 阶段成功实现了数据库层的高可用架构，以下是主要收获与经验：

1. MariaDB 主从复制配置需要注意 GTID 和二进制日志格式，确保数据一致性。
2. MaxScale 的监控用户需要精确的权限设置，确保能正确执行状态检查和故障切换。
3. 数据库用户授权时，必须考虑来源主机的 IP 地址，避免连接失败。
4. 遇到启动错误时候，检查日志是排查问题的关键。以后任何时候都要先检查日志。

2.4 L4 存储高可用实施

目标：保障存储服务在单节点故障下的持续可用性。
方案要点：

- 双节点存储：使用 DRBD 或 GlusterFS 副本卷配置。
- 故障模拟：断网或关机测试节点容错能力。
- 文件一致性验证：新上传文件能正确存储并同步。

IP 规划：新增 NFS IP 为 154.64.254.91
架构图：

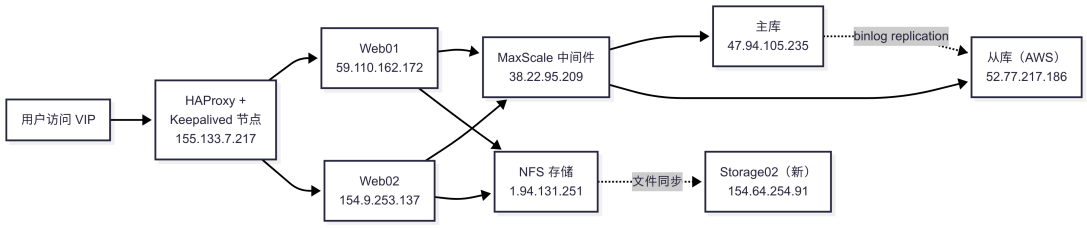

图 12: L4 存储高可用架构图

2.4.1 实操步骤

1. 环境准备

   两台存储节点（NFS01 和 NFS02）安装配置：

命令 *39*: NFS 节点初始化

```
# 在 nfs02 (154.64.254.91) 上执行
hostnamectl set-hostname nfs02
dnf update -y
dnf install -y nfs-utils glusterfs-server vim wget net-tools
   firewalld
systemctl enable --now firewalld
firewall-cmd --add-service=nfs --permanent
firewall-cmd --add-service=glusterfs --permanent
firewall-cmd --reload

# 在 nfs01 (1.94.131.251) 上执行相同配置
hostnamectl set-hostname nfs01
# 重复上述安装步骤
```

2. 部署 GlusterFS 分布式存储

   启动 GlusterFS 服务并建立集群：

命令 *40*: GlusterFS 集群配置

```
# 两台节点都执行
systemctl enable --now glusterd

# 在 nfs01 上执行
gluster peer probe 154.64.254.91
gluster peer status

# 创建存储目录
mkdir -p /gluster/wp-shared
```

```
10
11        # 创建副本卷（在 nfs01 上执行）
12        gluster volume create wp-shared replica 2 \
13        1.94.131.251:/gluster/wp-shared \
14        154.64.254.91:/gluster/wp-shared force
15
16        gluster volume start wp-shared
17        gluster volume info wp-shared
```

3. 配置 Web 节点挂载 GlusterFS

   在两台 Web 节点上安装 GlusterFS 客户端：

命令 *41*: Web 节点挂载 GlusterFS

```
1         # Web01 和 Web02 都执行
2         dnf install -y glusterfs-fuse
3
4         # 卸载原有 NFS 挂载
5         umount /var/www/html/wordpress/wp-content/uploads
6
7         # 挂载 GlusterFS 卷
8         mount -t glusterfs 1.94.131.251:wp-shared /var/www/html/
              wordpress/wp-content/uploads
9
10        # 设置永久挂载
11        vim /etc/fstab
12        # 修改原有行为:
13        1.94.131.251:wp-shared /var/www/html/wordpress/wp-content/
              uploads glusterfs defaults,_netdev 0 0
14
15        # 验证挂载
16        df -h | grep uploads
17        chown -R nginx:nginx /var/www/html/wordpress/wp-content/
              uploads
```

4. 测试存储高可用性

   - 在 WordPress 后台上传测试图片，验证文件正常存储。
   - 关闭 NFS01 节点，检查 WordPress 文件上传功能是否正常。
   - 重启 NFS01 节点，验证数据同步恢复功能。

命令 *42*: 故障测试

```
# 在 nfs01 上模拟故障
systemctl stop glusterd

# 在 Web 节点测试文件操作
echo "test-$(date)" > /var/www/html/wordpress/wp-content/
    uploads/test.txt

# 恢复 nfs01 服务
systemctl start glusterd

# 验证数据同步
gluster volume heal wp-shared info
```

2.4.2 问题解决记录

GlusterFS 卷创建失败：

在创建 GlusterFS 副本卷时遇到错误：`volume create: wp-shared: failed: Host 154.64.254.91 is not in 'Peer in Cluster' state`。

问题分析：检查防火墙设置和网络连通性后，发现 GlusterFS 服务未在所有节点正确启动。

解决方案：

命令 *43*: 重新初始化 GlusterFS

```
# 两台节点重新启动服务
systemctl restart glusterd

# 清除旧的对等关系
gluster peer detach 154.64.254.91 force

# 重新建立信任关系
gluster peer probe 154.64.254.91
gluster peer status
```

文件权限问题：

GlusterFS 挂载后，WordPress 无法写入文件，提示权限不足。

解决方案：

命令 *44*: 修复文件权限

```
# 在两台存储节点执行
```

```
    chown -R nobody:nobody /gluster/wp-shared
    chmod -R 755 /gluster/wp-shared

    # 在 Web 节点重新设置权限
    chown -R nginx:nginx /var/www/html/wordpress/wp-content/uploads
```

2.4.3 总结与反思

L4 阶段成功实现了存储层的高可用架构，主要收获包括：

1. GlusterFS 提供了良好的数据副本和故障切换能力，相比单节点 NFS 提升了可靠性。
2. 分布式存储的权限管理比传统 NFS 更复杂，需要在多个层面设置正确权限。
3. 故障测试验证了存储高可用的有效性，单节点故障不影响 WordPress 文件操作。
4. 建议生产环境使用专用存储网络，避免与业务流量混合。

3 总结与反思

3.1 项目成果总结

本次云计算高可用架构实战从 L1 到 L4 四个阶段逐步推进，成功构建了一套完整的高可用 WordPress 部署方案。各阶段核心成果如下：

- **L1 基础架构**：建立了 Web + DB + 存储的三层分离架构，验证了 LNMP 技术栈的基础部署能力。
- **L2 Web 层高可用**：通过 HAProxy + Keepalived 实现了前端服务的负载均衡与故障自动切换，消除了 Web 层单点故障。
- **L3 数据库高可用**：基于 MariaDB 主从复制和 MaxScale 代理，构建了数据库层的读写分离与自动故障切换机制。
- **L4 存储高可用**：采用 GlusterFS 分布式存储替代单节点 NFS，实现了文件数据的副本保护与故障恢复。

3.2 技术经验总结

3.2.1 部署配置方面

1. **权限管理一致性**：Web 服务器中 NGINX 与 PHP-FPM 的运行用户必须统一，避免文件写入权限冲突。生产环境建议创建专用的 `www` 用户组。

2. **数据库用户授权精确化**：MySQL/MariaDB 的权限模型基于 *用户名 @ 来源主机* 的组合，必须为每个访问来源（Web 节点、MaxScale 代理等）单独授权，避免连接失败。

3. **配置文件优先级**：NGINX 等服务的配置文件加载顺序影响最终效果，需要检查 `default_server` 等默认配置是否与自定义配置冲突。

4. **防火墙与安全组统一管理**：云环境下需要同时配置系统级防火墙（如 `firewalld`）和云平台安全组，确保端口访问策略一致。

3.2.2 故障排查方法

1. **日志优先原则**:遇到服务异常时,首先查看相关组件的日志文件(如 `/var/log/nginx/error.log`、`/var/log/mariadb/mariadb.log`)，日志信息往往能直接定位问题根因。

2. **分层测试法**：复杂架构问题应按层次逐一验证，如先测试数据库连接、再验证 Web 服务、最后检查负载均衡，避免多层次问题交织。

3. **网络连通性检查**：使用 `telnet`、`curl` 等工具验证端口可达性和服务响应，排除网络层面的干扰因素。

4. **权限问题排查**：文件操作失败时，通过 `ls -l`、`ps aux` 等命令核实文件属主与进程用户的匹配关系。

3.3 遇到的主要挑战

3.3.1 技术挑战

1. **MaxScale 权限配置复杂性**：MaxScale 的监控用户需要精确的权限组合（如 `SLAVE MONITOR`、`READ_ONLY ADMIN`），权限不足或过度都会影响故障切换效果。

2. **移动端浏览器安全策略**：现代浏览器对 HTTP + IP 地址访问的安全限制导致部分测试失败，需要通过域名绑定解决。

3. **GlusterFS 集群初始化**：分布式存储的对等关系建立和卷管理相对复杂，需要深入理解其工作机制。

4. **数据库管理**：数据库用户授权的方法和权限较为复杂。

3.4 学习成果与能力提升

通过本次实战项目，在以下几个方面获得了显著的能力提升：

- **系统架构设计**：掌握了从单节点到高可用架构的演进思路，理解了各层次高可用技术的适用场景。

- **Linux 系统管理**：深化了对 CentOS Stream 系统的配置管理、服务部署、故障排查等核心技能。
- **数据库管理**：熟练掌握了 MariaDB 的安装配置、主从复制、用户权限管理等关键操作。
- **网络服务配置**：具备了 NGINX、HAProxy、Keepalived 等网络服务的配置优化能力。
- **存储系统运维**：了解了从传统 NFS 到分布式存储（GlusterFS）的过程。
- **问题分析解决**：培养了系统性的故障排查思维和快速定位问题根因的实践能力。