线上故障排查标准流程 Checklist & 常用命令速查表

> 版本：v3.2 | 适用：Linux 运维 / SRE / 后端开发
> 基于 15 年生产环境实战经验总结，融合 SRE、Google SRE Book、Netflix Chaos Engineering 思想

第一部分：线上故障排查标准流程 Checklist

📋 总览：故障处理 5 个阶段

```
发现 → 响应 → 定位 → 恢复 → 复盘
 ↑_____________________________|
        持续改进闭环
```

阶段一：故障发现与告警接收（黄金 5 分钟）

1.1 告警接收 Checklist

- ☐ **确认告警真实性**：是真实故障还是误报？检查监控大盘、相关指标
- ☐ **查看告警上下文**：
  - 告警时间点
  - 影响范围（全局/局部/单实例）
  - 告警级别（P0/P1/P2）
  - 触发阈值与持续时间
- ☐ **建立事件频道**：拉群、打开 War Room（腾讯会议/Zoom 专用频道）
- ☐ **通知相关方**：oncall、上游业务方、客服

1.2 故障定级标准（建议）

级别	影响范围	响应时间	处理时长
P0	全站不可用/核心业务中断	5 分钟	30 分钟内恢复
P1	核心功能降级	15 分钟	2 小时内恢复
P2	非核心功能异常	30 分钟	8 小时内恢复
P3	轻微问题/体验问题	2 小时	24 小时内恢复

阶段二：信息收集与初步判断（5-15 分钟）

2.1 第一时间必看大盘（30 秒内）

```
# ==== 1. 全局健康度 ====
# 看监控大盘（Grafana/夜莺/Prometheus）
- 业务核心指标：QPS、RT、错误率、在线用户数
- 系统指标：CPU、内存、磁盘、网卡流量

# ==== 2. 核心服务存活 ====
curl -I http://service-name/health            # HTTP 健康检查
systemctl status nginx mysqld redis           # 服务状态
systemctl list-units --type=service --state=failed
```

2.2 故障黄金 30 秒判断表

现象	优先怀疑方向
5xx 错误率突增	上游服务故障 / 自身进程崩溃 / 配置错误
4xx 错误率突增	客户端问题 / 鉴权服务 / 参数校验
RT 飙升但无错误	DB 慢查询 / 锁等待 / 下游慢 / GC
CPU 飙高	死循环 / 流量上涨 / 慢调用累积 / 加密计算
内存飙高	内存泄漏 / 大对象 / 缓存过大 / OOM
磁盘满	日志暴增 / 数据写入 / 临时文件未清理
网络异常	带宽打满 / DNS / 防火墙 / 网卡丢包
服务连接不上	进程挂掉 / 端口未监听 / 安全组 / 防火墙

阶段三：分层排查（自顶向下 / 自底向上）

🔥 黄金法则：先恢复后定位，先止血后追因

3.1 网络层排查

```
# 网络连通性
ping <ip>                     # 基础连通性
traceroute <ip>               # 路由追踪
mtr -rwbzc 100 <ip>           # 持续追踪丢包率（推荐）
telnet <ip> <port>            # TCP 端口测试
nc -zv <ip> <port>            # nc 端口测试
curl -v http://<ip>:<port>    # HTTP 详细测试
curl -w "@curl-format.txt" -o /dev/null -s http://...  # 输出耗时详情

# DNS 排查
nslookup <domain>
dig <domain> +trace
```

```
cat /etc/resolv.conf

# 抓包分析（终极武器）
tcpdump -i eth0 -nn -s0 host <ip> and port <port> -w /tmp/cap.pcap
tcpdump -i any -nn -A port 80 | grep -i 'password'

# 查看连接状态
ss -s                           # 汇总统计
ss -tan state established        # 详细连接
netstat -an | grep ESTABLISHED | wc -l  # 已建立连接数
```

3.2 系统层排查

```
# ==== CPU ====
top -c                          # 实时 CPU（看 load average）
uptime                          # 负载情况
mpstat -P ALL 1                 # 每核 CPU
vmstat 1                        # 系统整体（含 IO）
# 排查高 CPU 进程
top -Hp <pid>                   # 看进程内线程 CPU
jstack <pid>                    # Java 线程栈
perf top -g                     # 性能分析神器
# 排查 CPU 飙高脚本
ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head

# ==== 内存 ====
free -h                         # 内存总览
cat /proc/meminfo               # 详细内存
# 排查高内存进程
ps aux --sort=-%mem | head
pmap -x <pid> | sort -k3 -n -r | head  # 进程内存映射
jmap -heap <pid>                # Java 堆详情
jmap -histo:live <pid> | head  # Java 对象统计

# ==== 磁盘 ====
df -h                           # 磁盘使用
df -i                           # inode 使用（重要！inode 满也会写不进去）
du -sh /* | sort -hr | head   # 大目录排查
ls -lhS /var/log/               # 大日志文件
# 磁盘 IO
iostat -xz 1                    # 磁盘 IO 统计
iotop                           # 进程级 IO
# 排查谁在写磁盘
lsof +L1                        # 被删除但还占空间的句柄（磁盘满杀手）

# ==== 网络流量 ====
iftop -i eth0                   # 实时流量
nethogs                         # 进程级流量
sar -n DEV 1                    # 网卡历史
ss -i                           # 网络连接详细信息
```

3.3 应用层排查

```
# ==== 进程状态 ====
ps auxf                          # 进程树
pstree -ap                       # 进程树状图
ls /proc/<pid>/                  # 进程内部信息
cat /proc/<pid>/status           # 进程状态
cat /proc/<pid>/limits           # 资源限制
cat /proc/<pid>/fd | wc -l     # 句柄数（排查句柄泄漏）

# ==== 句柄耗尽排查（Too many open files）====
ulimit -n
lsof -p <pid> | wc -l          # 进程打开的文件数
lsof | awk '{print $2}' | sort | uniq -c | sort -rn | head  # 按 PID 统计
# /etc/security/limits.conf 调大
* soft nofile 65535
* hard nofile 65535

# ==== 端口监听 ====
ss -tlnp                         # 监听的 TCP 端口
ss -ulnp                         # 监听的 UDP 端口

# ==== 应用日志（最重要的线索）====
tail -f /var/log/app.log
tail -1000 /var/log/app.log | grep -i error
grep -E "ERROR|FATAL|Exception" app.log | tail -50
# 日志上下文（关键时间点前后）
grep -B5 -A20 "ERROR" app.log
# JSON 日志查询（jq）
cat app.log | jq 'select(.level=="ERROR")' | tail -20
cat app.log | jq 'select(.timestamp > "2024-01-01T10:00:00" and .level=="ERROR")'
```

3.4 中间件层排查

```
# ==== MySQL ====
mysql -e "show processlist" | grep -v Sleep
mysql -e "show engine innodb status\G"   # InnoDB 状态
mysql -e "show slave status\G"           # 主从状态
# 慢查询
mysql -e "select * from information_schema.processlist where time > 5;"
# 锁等待
mysql -e "SELECT * FROM information_schema.INNODB_LOCKS;"
mysql -e "SELECT * FROM information_schema.INNODB_LOCK_WAITS;"

# ==== Redis ====
redis-cli info clients           # 客户端连接
redis-cli info memory            # 内存使用
redis-cli info stats             # 统计
redis-cli --bigkeys              # 大 key
redis-cli --latency              # 延迟
redis-cli slowlog get 100        # 慢命令
redis-cli config get maxmemory # 配置
```

```
# ==== Kafka ====
kafka-topics.sh --describe --topic xxx
kafka-consumer-groups.sh --describe --group xxx
# 消费 lag 监控

# ==== Nginx ====
nginx -t                          # 配置检查
tail -f /var/log/nginx/error.log
awk '{print $1}' access.log | sort | uniq -c | sort -rn | head  # TOP IP
awk '$9 == 500 {print $7}' access.log | sort | uniq -c | sort -rn  # 500 路径
```

3.5 K8s 环境特有排查

```
# ==== Pod 状态 ====
kubectl get pods -A -o wide
kubectl describe pod <pod>
kubectl logs <pod> --previous --tail=200
kubectl logs -f <pod> -c <container>
kubectl exec -it <pod> -- /bin/bash

# ==== 事件追踪 ====
kubectl get events --sort-by='.lastTimestamp' -A
kubectl get events -A --field-selector type=Warning

# ==== 资源/调度 ====
kubectl top node
kubectl top pod -A
kubectl describe node <node>  # 资源压力

# ==== 网络 ====
kubectl get endpoints
kubectl get svc
# 进入 pod 网络空间排查
kubectl debug <pod> -it --image=nicolaka/netshoot
```

阶段四：恢复操作（止血优先）

4.1 恢复原则

原则	说明
先恢复后定位	业务优先，快速止血
可回滚原则	任何变更必须有回滚方案
影响最小化	优先扩容/重启/降级，最后回滚
双人复核	关键操作必须二人确认

4.2 常见止血手段

```
# 1. 重启大法（70% 问题能解决）
systemctl restart <service>
supervisorctl restart <service>
kubectl rollout restart deployment/<name>

# 2. 扩容（应对流量上涨）
kubectl scale deployment/<name> --replicas=20
# 云上机器：控制台/API 扩容

# 3. 回滚（变更导致）
kubectl rollout undo deployment/<name>
git revert <commit> && 重新发布

# 4. 摘流（紧急止血）
# Nginx: upstream 中剔除问题节点
# LVS/API Gateway: 切流量到备用机房
# 配置中心：临时禁用某功能开关

# 5. 限流降级
# Sentinel/Hystrix: 配置降级规则
# 关闭非核心功能

# 6. 清理资源
# 清日志、删临时文件、清理大文件
> /var/log/big.log                # 截断日志（不要删！会断 inode）
# 删 7 天前日志
find /var/log -name "*.log" -mtime +7 -delete
# 清理缓存
sync && echo 3 > /proc/sys/vm/drop_caches

# 7. 紧急修复参数
sysctl -w net.core.somaxconn=65535
ulimit -n 65535
echo 1 > /proc/sys/vm/overcommit_memory  # Redis 推荐
```

阶段五：故障复盘（Postmortem）

5.1 复盘文档模板

```
# 【P1】xxx 故障复盘报告

## 1. 故障概述
- 故障时间：2024-01-15 10:00 ~ 11:30（90 分钟）
- 故障影响：下单接口 5xx 率 30%，影响 GMV 约 xxx 万
- 故障等级：P1

## 2. 时间线（Timeline）
- 09:50 - 研发 A 上线订单服务 v2.3.1
- 10:00 - 监控告警 5xx 率超阈值
```

```
- 10:02 - Oncall 收到告警
- 10:15 - 定位到 v2.3.1 引入 bug
- 10:20 - 执行回滚
- 10:25 - 服务恢复
- 11:30 - 全量恢复

## 3. 根因分析（5 Whys）
1. Why: 5xx 率飙升 → 代码 bug
2. Why: 出现 bug → 未充分测试
3. Why: 未充分测试 → 发布流程缺少灰度
4. Why: 缺少灰度 → 没有强制发布规范
5. Why: 没有规范 → 流程建设缺失

## 4. 改进措施（Action Items）
- [ ] 责任人: 张三, 截止: 2 月 1 日, 完成全链路灰度发布 SOP
- [ ] 责任人: 李四, 截止: 1 月 30 日, 补充单元测试覆盖率达 80%
- [ ] 责任人: 王五, 截止: 2 月 15 日, 上线 Chaos Engineering 演练

## 5. 经验教训
- ✅ 优点：响应及时，10 分钟内止血
- ❌ 缺点：缺乏灰度机制，全量上线风险大
```

5.2 复盘原则（Blameless Postmortem）

- ✅ 对事不对人，聚焦系统改进
- ✅ 假设每个人的决策在当时都是合理的
- ✅ 关注流程、机制，而非个人失误
- ❌ 禁止甩锅文化

第二部分：常用命令速查表

🔧 一、系统基础信息

命令	用途	关键参数
`uname -a`	系统内核信息	
`hostname`	主机名	
`uptime`	运行时长/负载	
`date`	当前时间	`date -d "yesterday"`
`who` / `w`	在线用户	
`last`	登录历史	`last -n 10`
`history`	命令历史	`history | grep xxx`
`dmidecode -t processor`	CPU 硬件信息	

命令	用途	关键参数
`lscpu`	CPU 架构信息	
`lsblk`	块设备列表	
`lspci`	PCI 设备	
`lsusb`	USB 设备	
`fdisk -l`	磁盘分区	

💾 二、CPU & 内存

CPU 排查

```
# 实时 CPU
top -c                                  # 进程视图
htop                                    # 增强版 top
mpstat -P ALL 1                         # 每核使用率
vmstat 1 10                             # 系统整体

# 历史 CPU
sar -u 1 10                             # 当前
sar -u -f /var/log/sa/sa15              # 历史

# 高 CPU 进程定位
ps -eo pid,user,pcpu,pmem,comm --sort=-pcpu | head
top -Hp <pid>                           # 进程内线程

# CPU 亲和性
taskset -cp <pid>                       # 查看 CPU 亲和性
taskset -c 0,1 <command>                # 绑定 CPU

# 性能分析（神器）
perf top -g                             # 函数级 CPU 热点
perf record -g -p <pid> -- sleep 30
perf report
```

内存排查

```
# 总览
free -h
cat /proc/meminfo

# 进程内存
ps aux --sort=-%mem | head
pmap -x <pid> | sort -k3 -n -r | head 20

# 内存详情
cat /proc/<pid>/status
```

```
cat /proc/<pid>/smaps | grep -E "Size|Rss|Pss"

# 缓存清理（紧急）
sync; echo 1 > /proc/sys/vm/drop_caches  # 清理 pagecache
sync; echo 2 > /proc/sys/vm/drop_caches  # 清理 dentries 和 inodes
sync; echo 3 > /proc/sys/vm/drop_caches  # 全部清理

# OOM 日志
dmesg | grep -i "out of memory"
grep -i "oom" /var/log/messages
journalctl -k | grep -i "oom"
```

三、磁盘 & IO

```
# 容量
df -h                                   # 人类可读
df -i                                   # inode
du -sh *                                # 当前目录占用
du -sh /* | sort -hr | head -10         # 根目录 TOP 10

# IO 性能
iostat -xz 1                            # 关键：%util await svctm
iotop -o                                # 进程级 IO
ionice -c 1 -n 0 <pid>                  # 调整 IO 优先级

# 大文件查找
find / -size +1G -exec ls -lh {} \;     # 大于 1G 的文件
find / -type f -mtime -1                # 1 天内修改的文件
find / -type f -mtime +30 -delete       # 删除 30 天前文件

# 文件删除但未释放（磁盘满杀手）
lsof +L1                                # 显示被删除但仍占空间的文件
# 找到 PID 后，根据进程决定重启或优雅处理

# 文件系统
mount                                   # 挂载点
mount \| column -t
/etc/fstab                              # 挂载配置
ls -la /sys/block/                      # 块设备

# RAID
cat /proc/mdstat                        # 软件 RAID
megacli -LDInfo -Lall -aAll             # 硬件 RAID
```

🌐 四、网络

网络配置

```
ip addr                                  # IP 地址（替代 ifconfig）
ip link                                  # 网卡状态
ip route                                 # 路由表
ip rule                                  # 策略路由
cat /etc/resolv.conf                     # DNS 配置
cat /etc/hosts                           # 静态解析
ethtool eth0                             # 网卡详细信息
```

网络流量

```
iftop -i eth0                            # 实时流量
nethogs                                  # 进程级流量
nload                                    # 简易流量图
sar -n DEV 1                             # 历史流量
sar -n TCP 1                             # TCP 统计

# 多机流量
iptraf-ng                                # 综合网络监控
```

网络连接

```
ss -s                                    # 汇总
ss -tan state established                # 已建立
ss -tan state time-wait \| wc -l         # TIME_WAIT 数
ss -ulnp                                 # UDP 监听
ss -tlnp                                 # TCP 监听

# 高并发排查
ss -tan \| awk '{print $5}' \| cut -d: -f1 \| sort \| uniq -c \| sort -rn | head
```

网络诊断

```
ping -c 5 -i 0.5 <ip>                    # 连通性
traceroute <ip>                          # 路由追踪
mtr -rwbzc 100 <ip>                      # 持续追踪
telnet <ip> <port>                       # TCP 测试
nc -zv <ip> 1-1024                       # 端口扫描
nc -ulnvp 9999                           # 监听 UDP 端口

# 域名解析
nslookup <domain>
dig <domain>
dig <domain> +trace                      # 完整解析路径
dig <domain> +short                      # 只看结果
host <domain>
```

```
# HTTP 测试
curl -v http://...                          # 详细
curl -I http://...                          # 头部
curl -w "time_total:%{time_total}\n" -o /dev/null -s http://...
curl -X POST -d 'data' http://...           # POST
curl -H "Content-Type: application/json" -X POST -d '{}' http://...

# 测速
iperf3 -c <ip>                              # 带宽测试
speedtest-cli                               # 公网测速

# 抓包
tcpdump -i eth0 -nn port 80                 # 抓 HTTP
tcpdump -i eth0 -nn -s0 -w /tmp/cap.pcap host 1.2.3.4
tcpdump -i any -nn -A 'port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420'
# 抓包转 HTTP（找 GET/POST）
```

🔥 五、进程管理

```
# 进程列表
ps aux
ps -ef
ps -eo pid,ppid,user,start,etime,pcpu,pmem,args --sort=-pcpu

# 进程树
pstree -ap
ps auxf

# 进程信息
cat /proc/<pid>/status
cat /proc/<pid>/cmdline | tr '\0' ' '
ls -la /proc/<pid>/exe                      # 可执行文件
ls -la /proc/<pid>/cwd                      # 工作目录
cat /proc/<pid>/environ | tr '\0' '\n'      # 环境变量

# 进程句柄
ls /proc/<pid>/fd | wc -l                   # 句柄数
lsof -p <pid>                               # 详细
cat /proc/<pid>/limits                      # 资源限制

# 信号
kill -l                                     # 信号列表
kill -15 <pid>                              # 优雅退出
kill -9 <pid>                               # 强制（最后手段）
killall -15 nginx                           # 按名字
pkill -f "pattern"                          # 按命令行

# 进程优先级
nice -n 19 <command>                        # 低优先级
renice -n 10 -p <pid>
```

```
# 守护进程
systemctl status <service>
systemctl restart <service>
journalctl -u <service> -f                # 服务日志
supervisorctl status
```

📜 六、日志分析

```
# 基础查看
cat <file>
less -N <file>                            # 带行号
tail -f <file>                            # 实时
tail -1000 <file> | grep "ERROR"          # 最近日志
head -100 <file>                          # 前 100 行

# 关键字搜索
grep -i "error" <file>                    # 不区分大小写
grep -E "ERROR|WARN" <file>               # 多关键字
grep -B5 -A20 "FATAL" <file>             # 上下文
grep -c "Exception" <file>                # 计数

# 高级搜索
awk '$9 == 500 {print $7}' access.log | sort | uniq -c | sort -rn
sed -n '/start/,/end/p' <file>            # 区间

# JSON 日志
cat app.log | jq '.'                      # 格式化
cat app.log | jq 'select(.level=="ERROR")'
cat app.log | jq -r 'select(.status>=500) | .url' | sort | uniq -c | sort -rn

# 大日志处理
# 千万别用 vim 打开 GB 级日志!
sed -n '1000000,2000000p' big.log > sample.log  # 切割
split -l 1000000 big.log part_                   # 切分

# 实时监控
multitail <file1> <file2>                 # 多窗口
lnav <file>                               # 增强版日志查看器

# 日志轮转
logrotate -d /etc/logrotate.conf          # 调试
cat /etc/logrotate.d/<service>
```

七、文本三剑客

```
# ==== grep ====
grep -i "error" file                      # 不区分大小写
grep -v "info" file                       # 反选
grep -E "ERROR|WARN" file                 # 扩展正则
```

```
grep -A5 -B5 "keyword" file              # 上下文
grep -c "pattern" file                   # 计数
grep -n "pattern" file                   # 行号
grep -r "pattern" /path/                 # 递归
grep -l "pattern" *.log                  # 只显示文件名

# ==== awk ====
awk '{print $1}' file                    # 第 1 列
awk -F: '{print $1}' /etc/passwd         # 指定分隔符
awk '$3 > 100 {print $0}' file           # 条件
awk '{sum += $1} END {print sum}' file   # 求和
awk 'NR==10,NR==20' file                 # 行范围
awk '{a[$1]++} END {for(k in a) print k,a[k]}' file  # 计数统计

# 经典案例
awk '{a[$1]++} END {for(k in a) print a[k],k}' access.log | sort -rn | head
ps aux | awk '{a[$3]+=1} END {for(k in a) print k"%:",a[k]}'

# ==== sed ====
sed -i 's/old/new/g' file                # 替换
sed -n '10,20p' file                     # 输出 10-20 行
sed -i '/^$/d' file                      # 删除空行
sed -i '1d' file                         # 删除第 1 行
sed -i '/pattern/d' file                 # 删除匹配行
```

🔍 八、文件 & 目录

```
# 查找
find / -name "*.log"                     # 按名字
find / -size +1G                         # 按大小
find / -mtime -1                         # 1 天内修改
find / -user <user>                      # 按用户
find / -type f -perm 777                 # 按权限
find / -name "*.log" -exec ls -lh {} \; # 查找并执行

# 软硬链接
ln -s /src /dst                          # 软链
ln /src /dst                             # 硬链

# 压缩解压
tar -czvf a.tar.gz /path/                # 压缩
tar -xzvf a.tar.gz                       # 解压
tar -tjvf a.tar.bz2                      # 查看
unzip a.zip
zip -r a.zip /path/

# 传输
scp file user@host:/path                 # 上传
scp user@host:/path/file .               # 下载
rsync -avz --progress src/ dst/          # 同步
rsync -avz -e ssh src/ user@host:/dst/   # SSH 同步
```

```
# 文件对比
diff file1 file2
vimdiff file1 file2

# 权限
chmod 755 file
chown user:group file
umask                                   # 当前掩码
```

🚀 九、性能分析大杀器

```
# ==== 综合 ====
# sysstat 工具包（强烈推荐安装）
sar -u 1 10                             # CPU
sar -r 1 10                             # 内存
sar -d 1 10                             # 磁盘
sar -n DEV 1 10                         # 网络
sar -b 1 10                             # IO
sar -q 1 10                             # 负载
sar -w 1 10                             # 上下文切换
# 历史查询
sar -u -f /var/log/sa/sa15
ls /var/log/sa/                         # 历史数据

# ==== strace（系统调用追踪）====
strace -p <pid>                         # 跟踪进程
strace -c -p <pid>                      # 统计系统调用
strace -T -e trace=network -p <pid>     # 网络调用
strace -f -e trace=open,read,write command  # 跟踪命令

# ==== ltrace（库调用追踪）====
ltrace -p <pid>

# ==== perf（性能分析神器）====
perf top                                # 实时热点
perf top -g                             # 调用栈
perf record -g -p <pid> -- sleep 30
perf report
perf stat <command>                     # 性能统计
perf list                               # 事件列表

# ==== bpftrace/eBPF（高级）====
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count(); }'

# ==== systemtap ====
stap -e 'probe syscall.* { printf("%s\n", probefunc()) }'
```

十、Docker & K8s

```
# ==== Docker ====
docker ps                                  # 运行中
docker ps -a                               # 全部
docker logs -f --tail 100 <container>      # 日志
docker exec -it <container> /bin/bash      # 进入
docker stats                               # 资源
docker top <container>                     # 进程
docker inspect <container>                 # 详情
docker system df                           # 磁盘使用
docker system prune                        # 清理
docker network ls                          # 网络

# ==== Kubernetes ====
# Pod
kubectl get pods -A -o wide
kubectl describe pod <pod>
kubectl logs <pod> -c <container> --previous
kubectl exec -it <pod> -- bash

# 部署
kubectl rollout status deployment/<name>
kubectl rollout history deployment/<name>
kubectl rollout undo deployment/<name>
kubectl scale deployment/<name> --replicas=10

# 网络
kubectl get svc,endpoints,ing
kubectl port-forward pod/<pod> 8080:80

# 调试（强烈推荐 netshoot 镜像）
kubectl run debug --rm -it --image=nicolaka/netshoot -- /bin/bash

# 资源
kubectl top node
kubectl top pod -A
kubectl describe node <node>

# 上下文
kubectl config get-contexts
kubectl config use-context <context>
```

十一、应急止血命令大全（收藏备用）

```
# ==== 救火专用 ====

# 1. 磁盘满
> /var/log/big.log                         # 截断日志（不要 rm）
find /var/log -name "*.log" -mtime +7 -delete
```

```
lsof +L1 | grep deleted                    # 找未释放空间

# 2. CPU 飙高
top -Hp <pid>                              # 定位线程
kill -3 <pid>                              # 打印 Java 栈（生成 thread dump）
kill -15 <pid>                             # 优雅重启
# 紧急：临时扩容
kubectl scale deployment/<name> --replicas=20

# 3. 内存飙高
sync; echo 3 > /proc/sys/vm/drop_caches  # 清缓存
# 紧急：重启
systemctl restart <service>

# 4. 句柄耗尽
ulimit -n 65535                            # 临时调大
echo "* soft nofile 65535" >> /etc/security/limits.conf

# 5. TIME_WAIT 过多
netstat -an | grep TIME_WAIT | wc -l
# 优化（应急）
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w net.ipv4.tcp_tw_recycle=0     # 注意：NAT 环境会出问题
sysctl -w net.ipv4.tcp_fin_timeout=10

# 6. 连接数打满
ss -s
# 查谁占的连接多
ss -tan | awk '{print $5}' | cut -d: -f1 | sort | uniq -c | sort -rn | head

# 7. 数据库连接池满
mysql -e "show processlist" | grep -v Sleep | wc -l
mysql -e "kill <id>"                       # 杀慢连接

# 8. Redis 内存满
redis-cli info memory
redis-cli FLUSHDB                          # 紧急清（慎用！）
redis-cli config set maxmemory-policy allkeys-lru

# 9. 网站 502/504
nginx -t                                    # 检查配置
tail -f /var/log/nginx/error.log
# upstream 摘除问题节点
sed -i 's/server 10.0.0.5/#server 10.0.0.5/' /etc/nginx/conf.d/upstream.conf
nginx -s reload

# 10. SSH 登录不上
# 走云控制台 VNC 登录
# 单用户模式
# /etc/ssh/sshd_config 检查 PermitRootLogin
systemctl restart sshd
```

🎯 十二、工具安装推荐

```
# CentOS/RHEL
yum install -y sysstat iotop iftop nethogs htop dstat \
    strace ltrace tcpdump nmap nc mtr multitail \
    lnav jq nload axel rsync

# Ubuntu/Debian
apt install -y sysstat iotop iftop nethogs htop dstat \
    strace ltrace tcpdump nmap netcat-openbsd mtr multitail \
    lnav jq nload axel rsync

# 推荐额外工具
# 1. ctop - 容器 top
sudo wget https://github.com/bcicen/ctop/releases/download/v0.7.7/ctop-0.7.7-linux-amd64 -O
/usr/local/bin/ctop
# 2. nethogs - 进程级流量
# 3. mycli - MySQL 增强
# 4. lnav - 日志查看器
# 5. fd - find 替代
# 6. bat - cat 替代（语法高亮）
# 7. tldr - 简化 man
```

📎 附录：黄金法则

🚨 故障处理 10 条军规

1. **先恢复，后定位** - 业务优先于排查
2. **可回滚** - 任何变更都有退路
3. **双人复核** - 关键操作二人确认
4. **保留现场** - 快照/日志备份后再操作
5. **沟通同步** - 及时告知业务方进展
6. **先撤后查** - 可疑操作先回滚，再排查
7. **避免连锁** - 操作前评估影响范围
8. **记录时间线** - 关键动作精确到分钟
9. **不甩锅** - blameless culture
10. **必有复盘** - 故障是最好的老师

📊 监控告警核心指标（4 大黄金指标）

```
流量 (Traffic)       - QPS / 请求量
错误 (Errors)        - 错误率 / 错误数
延迟 (Latency)        - P50/P95/P99 RT
饱和度 (Saturation)  - CPU/内存/磁盘/连接池使用率
```

> 💡 **最后建议**：
>
> 1. 收藏本文，遇到故障时对照 checklist 排查
> 2. 把常用命令做成 shell 别名或脚本（`alias diag='top -c'`）
> 3. 重要服务预先编写 runbook（运维手册）
> 4. 定期演练：Chaos Engineering、故障演练
> 5. 知识库沉淀：每次故障后完善团队 Wiki

文档版本：v3.2 | **最后更新**：2024-12
适用系统：CentOS 7+ / Ubuntu 18+ / RHEL 8+ / K8s 1.20+
反馈建议：运维无捷径，实践出真知 ✊