简历话术

（简历辅导+自我介绍+公司+项目）

传智播客

总部教学实施中心 云计算教研部

（第一版）

更新日志

版本	更新内容	更新时间
V1.0	1.完成初稿；	2025/04/13
V1.5	1.在【项目话术篇】部分，增加了信创项目；	2025/04/28
V1.6	1.在【项目话术篇】部分，增加了 ERP 项目；	2025/04/30
V1.7	1.在【公司话术篇】的【关于服务器规模】部分，增加了服务器型号的图文介绍；	2025/05/05
V1.8	1.在【项目话术篇】的【Shell 与 Linux 运维】部分，更新了两个项目常问（印象深刻的）问题；	2025/05/13
V1.9	1.在【前言】部分，增加了整体介绍；	2025/05/16
V1.10	1.增加【Shell 运维】部分面试题；	2025/06/14

前言

整本书分为四章，该如何使用呢?

第一章介绍了如何编写简历，依次包含基本信息、专业技能、公司履历、项目模板、项目描述等内容。除此之外，给出了一些经验分享和职业发展方向的介绍。对于每个工作经验（3 年内、3-5 年，5 年以上）阶段，分别给出了不同岗位的要求，将来同学们如果持续深耕 IT 行业，有助于把握技能要求和个人优势。

第二章介绍了如何进行自我介绍、公司介绍。这一章节，要求大家学习完，必须准备好 3-5 分钟的自我介绍，每天流畅背诵半个小时，坚持一星期，一定能够自信流利地和别人介绍自己。

第三章介绍了如何进行公司介绍。这一章节，对于没有实际工作经验的同学来说，大有帮助。我们会尽可能地补充一些大家关注或疑惑的点，进行图文并茂地介绍。

第四章介绍了如何进行项目介绍。这一章节，必须结合课程当中的项目，进行实战，在实战中总结项目经验。我们给出的面试必问问题，都是一定会被问到的题目。但不能靠背诵记忆来解决，必须结合项目，进行理解消化。如果发现问题回答不上来，一定要查漏补缺，分析项目中是不是漏掉了哪些总结内容。

最后，正文文档中，蓝色字体是模板和大纲，用来串联整体结构的，通读话术时可跳过。

目 录

第一章 简历辅导篇

1.1 基本信息

姓名：XXX　　工作经验：X 年　　+----------+

年龄：XX　　联系电话：XX　　| 个人 |

学历：本科　　电子邮箱：XX　　| 照片 |

籍贯：XX　　求职方向：运维　　+----------+

1.2 专业技能

在简历上撰写专业技能模板时，注意以下几点：

1.2.1 技能量化

- 把 “熟悉 Linux 相关操作”，丰富为 “熟悉 Linux 相关操作，3 年 Linux 运维经验”；
- 把“熟悉 K8S 集群”，丰富为“熟悉 K8S 集群，2 年 K8S 集群维护经验”；
- 有基础到进阶

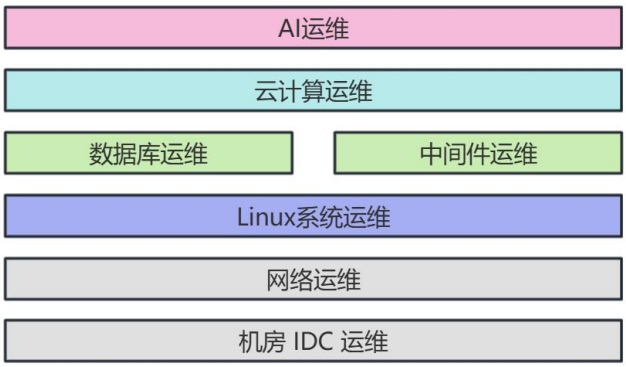
    1、Linux 基础

    2、数据库和中间件

    3、Docker 和 K8S 集群

    4、开源工具和云厂商产品

    5、行业持证



1.2.2 行业侧重

互联网大厂，看重 K8S、中间件；

中小公司，看重 Linux 系统运维、Python；

金融行业，数据安全和合规性是重点，看重数据库运维；

传统制造、医疗、电力、石油、交通、教育，看重 Linux 系统运维；

1.2.3 能力适当

5 年工作经验以下的，不写“精通”等词语；

3 年工作经验以下的，不写“全栈”等词语；

1.2.4 简洁精炼

不超过 8 条；每条不超过一行；

1.2.5 增加软技能

项目成本管控、团队管理协作、行业证书；

1.2.6 专有名词拼写

专有名词拼写正确，全文统一。

Linux、MySQL、Oracle、CentOS、Docker、Kubernetes、Python、Java、Zabbix、Ansible。比如，Linux，不能写成 linux。MySQL，不能写成 Mysql；

1.3 项目模板

项目模板包含两大块：**项目描述 + 个人职责**。

大家要注意，这里面项目描述和个人职责是**组合使用**的。老师重点给到大家的是个人职责的描述参考，项目描述自己结合熟悉或擅长的行业来写，比如，银行、煤矿、建筑、电商、物流等等。

1.3.1 注意事项

这一章节一定要看，这是提高邀约率的关键因素。

1.3.1.1 城市

- **小城市跳不到大城市：**想在北、上、深、杭找工作的，项目**不要写**在郑州、西安等城市的，因为技术栈不同；
- **学校和籍贯就近投递：**比如，家在山东，在东北上学，投递广州或长沙的岗位，必须给出充分的理由；

1.3.1.2 成长

- **行业聚焦：**在同一家公司，项目要连贯。如果今年做银行，明年突然上矿山，堆砌痕迹太明显；
- **成长连贯：**如果工作三年，最多两份工作，有一定成长轨迹。这两份工作可以略微跨行，但是项目必须连贯；

1.3.2 项目描述

1.3.2.1 银行

待定。

1.3.2.2 医院

待定。

1.3.2.3 煤矿

待定。

1.3.2.4 电信

待定。

1.3.2.5 电商

待定。

1.3.2.6 物流

待定。

1.3.3 个人职责

这一章节，不能全部抄。对于每个项目，确定下来自己的一个岗位角色，围绕这个岗位的个人职责，把握自己擅长的，从下面挑出来 4 到 6 条。即：

1. 每个项目，选定自己的一个岗位角色；
2. 选 5 条左右；

1.3.3.1 网络

- 负责交换机、路由器、防火墙等设备的初始化配置、日常维护及版本升级；
- 执行网络设备升级、链路切换等变更，执行回滚方案；
- 定期备份设备配置（自动化工具如 RANCID），核对 ACL/路由策略；
- 参与网络冗余测试（如主备链路切换），验证灾难恢复预案有效性；

1.3.3.2 Linux 系统

- 负责 Linux 服务器日常监控与巡检，及时发现并处理系统告警；
- 执行 Linux 系统安装、初始化配置及补丁更新，确保系统安全稳定运行；
- 管理用户账号、权限分配及密码策略，定期清理无效账号；
- 处理磁盘空间、内存、CPU 等资源不足问题，进行合理的资源规划与调整；
- 维护系统服务（如 SSH、HTTPD、Nginx 等），保障服务正常运行；
- 配置和优化网络参数（IP、路由、防火墙规则），保障服务器网络连通性；
- 使用 Shell 脚本或 Ansible 等工具实现系统管理任务自动化，提高运维效率；
- 定期备份重要数据和系统配置文件，验证备份的有效性和可恢复性；
- 排查系统故障，分析日志文件，定位并解决系统崩溃、服务异常等问题；
- 管理软件包安装、升级与卸载，解决依赖冲突问题；
- 监控系统进程，处理异常进程，保障系统性能；
- 搭建和维护基础服务（如 DNS、DHCP），确保网络服务正常运行；
- 协助开发人员部署应用程序，配置运行环境；
- 记录系统运维操作过程和问题处理方案，形成知识文档；
- 配合安全团队进行系统安全加固，修复安全漏洞；
- 参与系统架构优化，提出性能提升建议和方案；
- 监控系统日志，及时发现安全攻击和异常行为；
- 配合团队完成系统迁移、扩容等项目任务；
- 研究新技术和工具，应用于实际运维工作，提升运维水平；
- 协助制定和完善系统运维流程和规范，保障运维工作标准化；

1.3.3.3 MySQL 数据库

- 负责 MySQL 数据库安装、初始化配置，确保实例正常运行；
- 管理数据库用户账号，分配权限，定期审查账号权限有效性；
- 进行数据库备份与恢复操作，使用 mysqldump、xtrabackup 等工具保障数据安全；
- 处理慢 SQL 问题，通过 EXPLAIN 等命令分析并优化查询语句；
- 维护数据库表结构变更，确保数据完整性与业务兼容性；
- 监控数据库磁盘空间、内存等资源使用情况，及时处理资源不足问题；
- 执行主从同步配置与维护，确保数据一致性与高可用性；
- 清理冗余数据，优化表空间，提高数据库存储效率；
- 排查数据库连接异常、服务宕机等故障，快速定位并解决问题；
- 定期清理与归档数据库日志，避免日志占用过多空间；
- 协助开发人员进行数据库设计评审，提供性能优化建议；
- 进行数据库版本升级测试与实施，保障升级过程平稳；
- 配置并管理数据库参数，根据业务需求优化性能；
- 记录数据库运维操作过程与问题处理方案，形成文档；
- 配合安全团队完成数据库安全加固，修复安全漏洞；
- 监控数据库锁等待、死锁等问题，及时处理并优化；
- 参与数据库架构优化，协助制定分库分表等方案；
- 研究 MySQL 新特性与工具，应用于实际运维工作；
- 制定数据库应急预案，定期进行灾难恢复演练；

1.3.3.4 Redis 数据库

- 负责 Redis 数据库的安装部署与初始化配置，确保服务正常启动；
- 运用 Redis-cli、Info 命令等工具，掌握内存、CPU 等资源使用情况；
- 管理 Redis 用户权限，配置访问控制，保障数据访问安全；
- 制定并执行数据备份策略，使用 RDB、AOF 等方式完成备份与恢复操作；
- 排查 Redis 连接超时、服务中断等故障，快速定位并解决问题；
- 优化 Redis 内存使用，处理内存碎片，避免因内存不足导致服务异常；
- 配置主从复制、哨兵模式，保障数据高可用性与一致性；
- 协助开发人员调试 Redis 相关代码，解决缓存穿透、雪崩等问题；
- 定期清理过期数据，释放内存资源，提升性能；
- 监控 Redis 慢日志，分析并优化低效命令，提高响应速度；
- 执行 Redis 版本升级，提前测试兼容性，确保升级平稳；
- 记录运维操作与问题处理过程，形成详细文档便于后续参考；

- 配合安全团队进行 Redis 安全加固，修复潜在安全漏洞；
- 管理 Redis 集群，添加、删除节点，保证集群稳定运行；
- 分析 Redis 数据结构使用场景，提出优化建议，提升存储效率；
- 处理 Redis 事务冲突，保障数据操作的原子性与正确性；
- 监控 Redis 客户端连接数，防止因连接过多影响服务性能；
- 参与 Redis 架构优化，协助设计缓存策略与架构扩展方案；
- 研究 Redis 新特性与周边工具，应用于实际运维工作；
- 制定 Redis 应急预案，定期演练，确保故障时快速恢复服务；

1.3.3.5 Nginx 中间件

- 负责 Nginx 服务器安装、配置与初始化，确保服务正常启动与运行；
- 执行 Nginx 日常巡检，监控服务状态，及时处理异常告警；
- 管理 Nginx 虚拟主机配置，配置域名绑定、SSL 证书，实现 HTTPS 访问；
- 进行反向代理与负载均衡配置，合理分配流量，提升系统可用性；
- 优化 Nginx 性能参数，如连接数、缓存策略，提高响应速度；
- 配置 Nginx 限流、限速策略，防止恶意请求与流量冲击；
- 处理 Nginx 日志，分析访问情况与错误信息，定位并解决问题；
- 排查 Nginx 服务无法访问、502/504 等错误，快速恢复服务；
- 协助开发人员部署应用，配置 Nginx 与后端服务的衔接；
- 定期备份 Nginx 配置文件，确保配置可恢复；
- 执行 Nginx 版本升级与补丁更新，提前测试兼容性；
- 配置 Nginx 的 URL 重写、rewrite 规则，满足业务需求；
- 监控 Nginx 资源占用，处理因高负载导致的性能下降问题；
- 记录 Nginx 运维操作过程与问题解决方案，形成文档；
- 配合安全团队进行 Nginx 安全加固，防止常见 Web 攻击；
- 管理 Nginx 的 upstream 配置，动态调整后端服务器权重；
- 配置 Nginx 的 gzip 压缩功能，减少带宽占用；
- 参与 Nginx 架构优化，提出高可用、高性能改进建议；
- 研究 Nginx 新特性与模块，应用于实际运维工作；

1.3.3.6 Shell

- 编写 Shell 脚本实现 Linux 系统日常任务自动化，如文件备份、日志清理；
- 开发脚本监控系统资源（CPU、内存、磁盘），自动触发告警；
- 管理用户账号及权限，通过脚本批量创建、修改、删除用户；
- 编写脚本执行软件包的安装、升级与卸载，处理依赖关系；

- 解析系统日志文件，用 Shell 脚本提取关键信息并统计分析；
- 实现网络配置自动化，通过脚本批量修改 IP、路由规则；
- 编写脚本处理磁盘空间不足问题，自动清理临时文件；
- 开发脚本完成数据库备份、压缩及传输至存储服务器；
- 利用 Shell 脚本监控服务状态，异常时自动重启服务；
- 编写脚本批量处理文本数据，如替换字符串、提取字段；
- 实现服务器初始化配置脚本化，提高部署效率；
- 编写脚本实现文件系统权限批量设置与检查；
- 开发脚本获取系统进程信息，处理异常进程；
- 编写脚本生成系统状态报告，输出关键指标；
- 实现 Shell 脚本与其他工具（如 curl、awk）结合完成复杂任务；
- 编写脚本自动化执行系统补丁更新流程；
- 开发脚本解析配置文件，提取并修改关键参数；
- 利用 Shell 脚本完成服务器间文件同步与分发；
- 编写脚本实现任务调度，结合 crontab 执行定时任务；
- 维护 Shell 脚本版本，优化脚本逻辑，提高执行效率；

1.3.3.7 自动化

- 使用 Ansible 工具编写 playbook，实现服务器配置自动化；
- 基于 Python 编写脚本，自动处理服务器监控告警，如资源不足时的自动扩容；
- 维护自动化运维工具的版本更新，确保工具稳定可用；
- 利用脚本批量创建、管理云主机实例，提高资源部署效率；
- 开发自动化备份脚本，定期备份重要数据和系统配置文件；
- 编写自动化测试脚本，对新部署的服务进行功能和性能验证；
- 优化自动化运维流程，减少人工干预，提升运维效率；
- 使用 Prometheus、Grafana 等工具实现监控告警自动化；
- 建立自动化日志分析系统，自动识别异常日志并告警；
- 通过自动化工具进行服务器软件包的批量安装、升级和卸载；
- 开发脚本自动清理过期数据和临时文件，释放磁盘空间；
- 配置自动化任务调度，合理安排定时任务执行时间；
- 协助开发人员进行自动化部署环境的搭建与配置；
- 编写脚本自动生成系统状态报告，输出关键指标；
- 维护自动化运维脚本版本，及时修复脚本中出现的问题；
- 利用自动化工具进行服务器安全基线检查与加固；
- 开发自动化脚本来处理服务器的日常巡检工作；

- 搭建自动化的灾备切换演练环境，定期验证灾备方案有效性；
- 学习新技术和工具，将其应用到自动化运维工作中；

1.3.3.8 容器

- 负责容器（Docker）的安装部署、镜像制作与管理，确保基础运行环境正常；
- 使用 Kubernetes 部署应用，管理 Pod、Service、Deployment 等资源；
- 执行容器日常监控，通过 Prometheus 等工具监测资源使用与运行状态；
- 处理容器镜像拉取失败、Pod 崩溃等故障，快速定位并解决问题；
- 维护容器网络配置，保障容器间及与外部的网络连通性；
- 管理容器存储，配置 PVC、PV，确保数据持久化存储；
- 制定并执行容器备份策略，保障应用数据可恢复；
- 优化容器资源分配，调整 CPU、内存限制，提升资源利用率；
- 协助开发人员进行容器化应用部署，解决环境适配问题；
- 定期清理无用容器与镜像，释放磁盘空间；
- 执行 Kubernetes 集群升级，提前测试并保障升级平稳；
- 配置容器的健康检查，自动处理异常容器的重启与替换；
- 记录容器运维操作与问题处理过程，形成知识文档；
- 配合安全团队进行容器安全加固，修复潜在漏洞；
- 监控容器日志，分析异常行为并及时响应；
- 管理容器资源配额，防止资源滥用影响集群稳定性；
- 参与容器集群架构优化，提出高可用改进建议；
- 研究容器新技术（如 K8s 新特性），应用于实际运维工作；
- 制定容器应急预案，定期演练确保故障快速恢复；
- 维护容器镜像仓库，管理镜像版本与权限控制；

1.3.3.9 K8s

- 负责 Kubernetes 集群安装部署，使用 kubeadm 或二进制方式搭建并初始化集群；
- 日常监控集群节点状态、资源使用情况，及时处理节点异常离线问题；
- 管理 Pod、Deployment、Service 等核心资源，进行应用的部署、更新与回滚操作；
- 配置网络插件（如 Calico、Flannel），保障集群内网络通信正常；
- 处理 Pod 创建失败、服务暴露异常等常见故障，通过日志定位并解决问题；
- 维护 StorageClass、PV、PVC，实现数据持久化存储与动态分配；
- 使用 HPA 实现应用的自动扩缩容，保障业务高可用；
- 协助开发人员进行 K8s 环境适配，解决容器化应用部署问题；
- 定期备份 K8s 核心组件配置，防止配置丢失；

- 执行 K8s 集群版本升级与补丁更新，提前测试兼容性；
- 配置 Ingress 资源，实现域名访问与流量分发；
- 监控集群日志，通过 EFK 或 ELK 等工具分析异常行为；
- 管理 K8s RBAC 权限，分配用户与服务账号权限；
- 优化集群资源调度策略，提升资源利用率；
- 记录 K8s 运维操作与问题处理过程，形成详细文档；
- 配合安全团队进行 K8s 安全加固，防范常见攻击；
- 处理集群资源配额超限问题，合理分配资源；
- 参与 K8s 集群架构优化，提出性能改进建议；
- 研究 K8s 新特性与工具（如 Operator），应用于实际工作；
- 制定 K8s 应急预案，定期演练保障故障快速恢复；

1.3.3.10 Prometheus

- 负责 Prometheus 的安装部署与基础配置，确保监控服务正常启动；
- 配置 Prometheus 采集任务，定义 Exporter 监控指标，对接服务器、数据库目标；
- 管理 Prometheus 规则，编写告警规则与记录规则，实现指标聚合与阈值判断；
- 日常监控 Prometheus 运行状态，查看内存、磁盘使用及任务采集延迟情况；
- 排查监控数据缺失、指标采集失败等问题，修复 Exporter 配置或网络故障；
- 维护 Prometheus 存储，配置数据保留策略，清理过期监控数据；
- 协助开发团队接入应用自定义监控指标，提供 Exporter 开发建议；
- 优化 Prometheus 查询性能，调整存储引擎参数，避免慢查询影响服务；
- 执行 Prometheus 版本升级与补丁更新，测试兼容性保障平滑过渡；
- 配置 Prometheus 多实例与联邦集群，实现大规模监控数据采集；
- 记录 Prometheus 运维操作与问题处理过程，形成标准化文档；
- 配合 Grafana 完成监控面板设计，提供指标数据支持；
- 管理 Prometheus 的身份认证与访问控制，保障数据安全；
- 监控 Prometheus 告警通道，确保告警信息及时准确推送；
- 处理 Prometheus 资源占用过高问题，调整硬件资源或优化配置；
- 定期验证告警规则有效性，模拟触发场景测试告警响应；
- 研究 Prometheus 新特性与插件，如远程存储、边车模式等；
- 制定 Prometheus 备份恢复方案，保障监控数据可恢复性；
- 优化监控数据采集频率与精度，平衡性能与资源消耗；
- 制定 Prometheus 应急预案，定期演练保障故障快速处置；

1.3.3.11 CI/CD

- 安装配置 Jenkins/GitLab CI/Drone 等 CI/CD 工具，确保服务稳定运行；
- 管理 GitLab/GitHub 代码仓库，配置 Webhook 触发自动化流程；
- 维护 Docker Registry，保障镜像存储与拉取高效稳定；
- 编写 Jenkinsfile/GitLab CI 配置，定义构建、测试、部署阶段；
- 配置 Maven/Gradle/npm 等构建工具，优化依赖缓存提升速度；
- 集成静态代码分析工具（SonarQube），执行代码质量检查；
- 实现自动化测试（单元测试、集成测试），确保代码质量；
- 配置 Kubernetes/VM 部署目标，实现应用容器化部署；
- 管理环境变量与密钥，通过 Vault/AWS Secrets 等工具安全注入；
- 实现蓝绿部署、金丝雀发布等策略，降低发布风险；
- 配置 Prometheus/Grafana 监控 CI/CD 流程指标（构建时长、成功率）；
- 设置告警规则，及时通知构建失败、资源异常等情况；
- 清理过期构建产物与镜像，释放存储资源；
- 优化流水线性能，减少不必要的步骤与等待时间；
- 维护插件生态，更新工具版本，修复安全漏洞；
- 排查构建失败、部署异常等问题，分析日志定位根因；
- 协助开发团队解决环境依赖与配置问题；
- 记录运维操作与解决方案，沉淀技术文档；
- 实施权限控制，确保不同角色访问最小化必要资源；
- 审计 CI/CD 流程，确保符合企业安全与合规要求；

1.4 经验分享

1.4.1 工作经验年限

为什么面试经验经常卡 3 年，5 年，而不是 2 年，4 年，6 年？

第一年熟悉运维环境，基本工具；

第二年参与项目运维，积累知识文档；

第三年独立承担运维模块，团队带新；

第五年时，着重提升【软技能】

在 1 家公司内走向部门技术老员工，或者转了项目管理岗；

在 2 家公司积累项目运维经验，熟悉行业生态；

第七年时（30-32 岁）

在公司相对稳定下来，平衡工作和家庭

1.4.2 运维人员规模

小型公司（<100 人）：5－10 名运维人员；

中型公司（100~500）：10－30 名运维人员；

大型公司（>500 人）：超过 50 名运维人员；

一个人直接管理的人数不超过 10 人；超过之后，会分组，安排运维小组长；比如，天津某家公司，40 个研发人员的团队，其中运维 14 人，4 个硬件、4 个硬件+网络、4 个软件；

1.4.3 运维职业规划

运维只是入行计算机领域的一个切入点。一般三年就会入行。

不断巩固 Shell；

根据公司业务，在 Python 或 Go 中选择一个熟练掌握；

运维人员对 Java 做到了解 JDK 和 JVM 区别即可；

每个行业都有一个垂直的生态，公司架构一般包含九大类：

销售：市场调研、客户扩展

商务：寻找合作，建立伙伴关系

运营：产品运营、内容运营、用户营运

设计：工业设计、平面设计、产品设计、解决方案

技术：硬件、软件(前端、后端、数据库)、算法、网工、网安、项目

生产：车间流水线、质量检验、仓储物流

财务：会计、出纳

法务：风控、合规

人力：招聘、培训、员工福利、员工激励

1.4.4 行业公司分类

计算机公司按**类型**包含：

- 国央企：科技子公司居多，其他行业的 IT 部门，技术稳定
- 私企：大型、中小型互联网，技术更迭快

- 人力外包：国央企卡学历，如果进到了相应的外包，节奏会比较慢，生活舒适。大型私企越来越多外包，查漏补缺，不要卷，不要幻想转正，攒钱

计算机公司按**人数**分为：

- 1000 人：属于大型公司，企业有成功经验，可以学到很多东西，公司文化取决于创始人性格
- 500 人：有稳定的产品和服务，节奏各样，二三线居多，不卡学历
- 200 人：处在盈亏平衡点，或融资阶段，不卡学历
- 50 人：初创居多，当下的 AI、机器人赛道可以投身
- 个体户：监控安装/电脑维修(本地生活)、打印店(淘宝网店)

未来计算机行业发展机会：

- 2030 年 - AI（大文本|专家模型、自动驾驶），短期烧钱，长期会成功；
- 2035 年 - 自动驾驶、具身智能、AGI 对工作的挑战；
- 2040 年 - 低空经济；
- 2050 年 - 航空航天；

在就业、跳槽的时候，结合【兴趣、薪酬、行业、城市】维度进行评分；

1.4.5 运维项目周期

自研项目：2 年

面向政企行业的项目（人派到项目）：

- 200w - 500w： 6-12 月；
- 50w - 100w：1-3 个月；
- 10w - 50w：一星期；

1.4.6 一线、二线、三线

运维面试里，提到的一线、二线、三线是什么？做什么的？

一线：

客服岗，实施岗；

响应客户的 400 电话，解答基础问题，转交复杂问题给二线，并记录工单；

一线说的最多的术语，就是——工单、记录、转派；

二线：

业务运维岗，实施岗；

分析一线转交的复杂问题（如性能瓶颈、集群故障），拉开发来定位；

三线：

开发岗，系统运维岗

系统 Java、Python 研发，系统运维工作；

总结：一线跟客户接触最近；二线是两头拉通；三线聚焦系统设计和研发；

1.5 职业细分模板

未来运维发展，除了看学历和履历，在技能方面，主要包含以下职责，供参考。

1.5.1 初级工程师（2 年内）

模板一：基础运维

1. 掌握 Linux 系统常用命令，可完成文件操作、用户管理，独立安装 CentOS 系统。
2. 了解 Shell 脚本基础，能编写简单脚本进行文件备份。
3. 知晓 Docker 基本概念，能在指导下启动和停止容器。
4. 熟悉 MySQL 基础操作，如创建数据库、表和插入数据。
5. 了解防火墙 iptables 规则，能设置简单访问策略。

模板二：网络与系统

1. 熟悉 TCP/IP 协议，能进行简单的网络配置和故障排查。
2. 掌握 Linux 系统服务管理，可启动、停止和重启常见服务。
3. 了解 Docker 容器网络模式，能进行基本的网络连通测试。
4. 知道 MySQL 数据库的备份方法，能使用简单工具进行备份。
5. 了解 Zabbix 监控系统，能查看基础监控指标。

模板三：存储与系统

1. 了解 NFS 网络存储，能进行简单的挂载和使用。
2. 掌握 Linux 系统磁盘管理，包括分区、格式化。
3. 知道 Docker 数据卷的概念，能进行数据卷挂载。
4. 熟悉 MySQL 数据库的数据目录结构。

5. 了解日志文件管理，能查看和清理系统日志。

模板四：容器与系统

1. 掌握 Docker 容器的基本操作，如创建、删除和查看容器状态。
2. 了解 Linux 系统的进程管理，能查看和终止进程。
3. 知道 MySQL 数据库的用户管理，能创建和分配用户权限。
4. 了解防火墙的基本功能，能开放和关闭指定端口。
5. 了解简单的自动化运维概念，能使用脚本执行简单任务。

模板五：数据库与系统

1. 熟练掌握 MySQL 数据库的基本操作，包括增删改查。
2. 掌握 Linux 系统的文件权限设置，保障数据安全。
3. 了解 Docker 镜像的拉取和保存方法。
4. 了解网络端口的概念，能查看端口占用情况。
5. 了解 Zabbix 监控的基本配置，能添加监控主机。

模板六：系统与监控

1. 掌握 Linux 系统的性能监控工具，如 top、vmstat。
2. 了解 Docker 容器的资源限制设置。
3. 熟悉 MySQL 数据库的配置文件，能进行简单修改。
4. 了解防火墙的日志记录功能，能查看访问日志。
5. 了解自动化运维工具 Ansible 的基本概念。

模板七：云服务基础

1. 了解阿里云或腾讯云等云平台的基本服务，如 ECS 云服务器。
2. 掌握云服务器的创建和登录操作。
3. 了解 Docker 容器在云平台上的部署方式。
4. 知道 MySQL 数据库在云平台上的使用方法。
5. 了解云平台的安全组配置，能设置基本规则。

模板八：应用部署

1. 熟悉 Apache 服务器的安装和配置，能部署简单静态网站。

2. 掌握 Linux 系统的环境变量设置，方便应用部署。
3. 了解 Docker 容器化应用的部署流程。
4. 知道 MySQL 数据库与应用的连接配置。
5. 了解应用服务的启动和停止方法。

模板九：网络与存储

1. 熟悉网络设备（如路由器、交换机）的基本配置。
2. 掌握 iSCSI 存储的基本使用，能连接存储设备。
3. 了解 Docker 容器的网络隔离机制。
4. 知道 MySQL 数据库的数据存储方式。
5. 了解网络拓扑结构，能进行简单的网络规划。

模板十：安全与运维

1. 了解 Linux 系统的安全加固方法，如用户密码策略。
2. 掌握 Docker 容器的安全管理，如镜像扫描。
3. 熟悉 MySQL 数据库的安全设置，如远程访问限制。
4. 了解防火墙的入侵检测功能，能设置简单规则。
5. 了解运维工作中的安全规范和流程。

1.5.2 中级工程师（3 到 5 年）

模板一：综合运维

1. 熟练掌握 Linux 系统运维，能进行性能调优、安全加固和故障排除。
2. 熟悉 Shell 脚本编写，可开发复杂自动化脚本实现系统监控和任务调度。
3. 熟练运用 Docker 进行镜像创建、管理和容器编排，使用 Docker Compose 管理多容器应用。
4. 熟悉 MySQL 数据库管理，能进行高可用架构设计、性能优化和数据备份恢复。
5. 熟练配置和管理防火墙 iptables，制定完善的网络安全策略。

模板二：容器与网络

1. 熟悉 Docker 和 Kubernetes 容器技术，能独立搭建和维护大规模集群。
2. 深入理解网络协议，能进行复杂网络配置和故障排查，优化网络性能。

3. 熟悉 MySQL 和 Redis 数据库的高级应用，如主从复制、集群搭建和缓存优化。
4. 熟练使用自动化运维工具（如 Ansible、Jenkins）搭建 CI/CD 流水线。
5. 熟悉 Zabbix 和 Prometheus 监控系统，建立全面的监控和预警体系。

模板三：存储与数据库

1. 熟悉 NFS、Ceph 等分布式存储系统的部署、配置和管理，实现高效数据存储。
2. 熟悉 MySQL、Oracle 等数据库的管理，能进行复杂的数据库架构设计和性能调优。
3. 熟悉 Docker 容器的存储管理，实现数据持久化和高效读写。
4. 熟悉自动化运维脚本开发，使用 Python 实现存储和数据库的自动化管理。
5. 了解数据库安全审计和容灾方案，保障数据安全和可用性。

模板四：云原生运维

1. 熟悉多种云平台（如阿里云、腾讯云、AWS）的架构和服务，能进行混合云架构设计。
2. 熟悉 Kubernetes 高级特性，如自定义资源定义、Operator 开发，实现复杂应用的自动化运维。
3. 熟悉微服务架构和实践，使用 Istio 进行服务治理和流量管理。
4. 熟练运用 CI/CD 工具实现云原生应用的快速迭代和交付。
5. 建立完善的云原生监控和日志管理体系，保障应用的稳定性和性能。

模板五：自动化运维

1. 熟悉 Ansible、SaltStack 等自动化运维工具，实现大规模服务器的自动化配置和管理。
2. 熟练编写 Python 脚本进行复杂的自动化任务开发，如批量软件安装和配置更新。
3. 熟悉 Docker 和 Kubernetes 容器的自动化部署和伸缩，实现资源的高效利用。
4. 熟悉 Jenkins、GitLab CI/CD 等持续集成和持续交付工具，搭建高效的开发流水线。
5. 建立自动化监控和报警机制，及时发现和解决系统问题。

模板六：网络安全

1. 深入理解网络安全技术，能进行网络渗透测试和漏洞修复，保障网络安全。
2. 熟悉防火墙、入侵检测系统（IDS）和虚拟专用网络（VPN）的配置和管理。
3. 熟悉 Docker 容器的安全加固技术，实现容器级别的安全防护。

4. 熟悉网络拓扑结构和路由协议，能进行网络优化和故障排除。
5. 了解安全审计和合规性要求，制定完善的安全策略和流程。

模板七：应用与数据库

1. 熟悉 Apache、Nginx、Tomcat 等应用服务器的部署、配置和优化，保障应用的高可用性。
2. 熟悉 MySQL、PostgreSQL 等数据库的管理，能进行数据库性能调优和故障排除。
3. 熟悉 Docker 容器化应用的部署和管理，实现应用的快速迭代和交付。
4. 熟悉应用性能监控工具（如 New Relic），优化应用性能。
5. 了解应用开发流程，能与开发团队紧密合作，保障应用的稳定运行。

模板八：监控与日志管理

1. 熟悉 Zabbix、Prometheus、Grafana 等监控工具，建立全面的系统监控体系。
2. 熟悉 ELK Stack、Graylog 等日志管理工具，实现日志的高效收集、分析和可视化。
3. 熟悉 Docker 容器和 Kubernetes 集群的监控和日志管理方法。
4. 运用数据分析和机器学习技术，从监控和日志数据中发现潜在问题和趋势。
5. 建立监控和日志管理的自动化流程，提高运维效率。

模板九：混合云与多集群运维

1. 熟悉混合云架构设计和部署，实现多云环境下的资源统一管理和调度。
2. 熟悉 Kubernetes 多集群管理技术，使用 Cluster API 等工具进行集群创建和管理。
3. 熟悉云原生存储和网络技术，实现混合云环境下的数据存储和网络通信。
4. 熟悉多云环境下的安全策略和合规性要求，保障混合云的安全运行。
5. 建立混合云环境下的监控和日志管理体系，及时发现和解决问题。

模板十：运维开发全栈

1. 具备全栈开发能力，熟悉前端（如 HTML、CSS、JavaScript）和后端（如 Python、Java）开发技术。
2. 熟悉 Docker 和 Kubernetes 容器技术，将开发的应用进行容器化部署和管理。
3. 熟悉自动化运维工具和 CI/CD 流程，实现开发和运维的无缝衔接。
4. 熟悉数据库设计和开发，能开发和管理数据库应用。
5. 具备系统架构设计和优化能力，设计高效、稳定的运维系统。

1.5.3 高级工程师（5 年以上）

模板一：高级运维架构师

1. 熟悉多种云平台（阿里云、腾讯云、华为云）架构，能独立设计和部署大规模混合云解决方案。
2. 熟悉 Kubernetes 集群的高级管理，包括集群扩展、升级、安全加固，应对复杂生产环境故障。
3. 设计和优化企业级数据库架构，如 MySQL 多主集群、Redis 分布式集群，保障数据高可用和高性能。
4. 构建完善的自动化运维体系，涵盖自动化部署、配置管理、监控报警，大幅提升运维效率。
5. 制定全面的网络安全策略，结合防火墙、入侵检测等技术，保障企业网络和应用安全。

模板二：云原生技术专家

1. 深入研究云原生技术栈，如 Istio 服务网格、Knative 无服务器计算，推动企业应用向云原生转型。
2. 熟悉 Kubernetes 高级特性，如自定义控制器开发、CRD 扩展，实现复杂业务场景的自动化运维。
3. 设计和实施云原生监控和日志管理方案，运用 Prometheus、Grafana、ELK 等工具，保障系统可观测性。
4. 带领团队进行云原生应用开发和部署，遵循 DevOps 理念，实现快速迭代和持续交付。
5. 与云服务提供商紧密合作，熟悉最新云原生技术动态，优化企业云资源使用。

模板三：数据库与存储专家

1. 熟悉多种数据库（Oracle、MySQL、PostgreSQL）的高级管理，包括性能调优、备份恢复策略制定、数据迁移。
2. 设计和部署企业级存储解决方案，如 Ceph 分布式存储、NetApp 存储阵列，保障数据高效存储和访问。
3. 建立数据库和存储的自动化运维体系，使用脚本和工具实现自动化监控、故障处理和资源分配。

4. 负责数据库和存储的安全管理，包括用户权限管理、数据加密、审计，确保数据安全合规。
5. 提供数据库和存储相关的技术支持和培训，提升团队整体技术水平。

模板四：网络与安全架构师

1. 设计和规划企业级网络架构，包括广域网、局域网、数据中心网络，保障网络高可用性和性能。
2. 构建多层次的网络安全防护体系，包括防火墙、入侵检测、加密技术，应对各种网络安全威胁。
3. 熟悉软件定义网络（SDN）和网络功能虚拟化（NFV）技术，实现网络的自动化和智能化管理。
4. 制定网络安全应急响应预案，在发生安全事件时迅速响应和处理，减少损失。
5. 与安全厂商和监管机构保持沟通，了解最新安全动态，更新企业安全策略。

模板五：自动化运维总监

1. 主导企业自动化运维体系建设，制定自动化运维战略和规划，推动运维工作的全面自动化。
2. 熟悉各类自动化运维工具（Ansible、SaltStack、Jenkins），开发和优化自动化脚本和流程。
3. 建立自动化运维监控和预警平台，实时监控系统状态，自动触发故障处理流程。
4. 带领团队进行运维流程优化，提高运维效率和质量，降低运维成本。
5. 与开发团队紧密合作，推动 DevOps 文化落地，实现开发和运维的深度融合。

模板六：容器技术专家

1. 熟悉 Docker 和 Kubernetes 容器技术的底层原理，能进行容器技术的定制开发和优化。
2. 设计和部署大规模容器集群，实现资源的高效利用和应用的高可用部署。
3. 建立容器安全管理体系，包括镜像安全、容器隔离、访问控制，保障容器化应用的安全运行。
4. 推动企业应用的容器化改造，制定容器化迁移方案，确保迁移过程的平稳过渡。
5. 与开源社区保持密切联系，参与容器技术的开源项目，提升企业在容器领域的技术影响力。

模板七：大数据运维专家

1. 熟悉大数据技术栈（Hadoop、Spark、Hive）的运维管理，保障大数据平台的稳定运行和高性能。
2. 设计和优化大数据存储和计算架构，提高数据处理效率和存储利用率。
3. 建立大数据监控和预警体系，实时监控大数据集群的性能和状态，及时处理故障。
4. 负责大数据安全管理，包括数据加密、访问控制、数据脱敏，保障数据安全和隐私。
5. 与数据科学家和分析师合作，提供大数据平台的技术支持，推动大数据业务发展。

模板八：混合云运维专家

1. 熟悉混合云架构和技术，能进行混合云环境下的资源调度、应用部署和管理。
2. 设计和实施混合云安全策略，保障混合云环境下的数据安全和网络安全。
3. 建立混合云监控和管理平台，实现对多云环境的统一监控和管理。
4. 解决混合云环境下的兼容性和性能问题，优化混合云架构和资源配置。
5. 与云服务提供商合作，制定混合云成本优化方案，降低企业云使用成本。

模板九：应用运维专家

1. 熟悉各类应用服务器（Apache、Nginx、Tomcat）的高级配置和优化，保障应用的高性能和高可用性。
2. 设计和实施应用性能监控和调优方案，使用性能分析工具（如 New Relic）提升应用响应速度。
3. 建立应用运维自动化体系，实现应用的自动化部署、升级和故障恢复。
4. 与开发团队紧密合作，参与应用开发过程，提供运维建议和技术支持。
5. 处理应用级别的安全问题，如漏洞修复、安全加固，保障应用的安全运行。

模板十：运维技术顾问

1. 为企业提供全面的运维技术咨询服务，包括架构设计、技术选型、流程优化等方面的建议。
2. 评估企业现有运维体系的优缺点，制定针对性的改进方案，提升企业运维水平。
3. 跟踪行业最新技术动态，为企业引入先进的运维技术和理念，推动企业技术创新。
4. 指导和培训企业运维团队，提升团队成员的技术能力和综合素质。
5. 参与企业重大项目的规划和实施，提供运维技术支持和保障。

第二章 自我介绍篇

按照基本信息、个人经历、项目介绍、离职原因的步骤介绍，如果还愿意听，介绍最近一份项目。整体不能低于 3 分钟。

2.1 基本信息

面试官你好，我叫 XXX，今年 XX 岁，很高兴参加今天的面试。在运维行业，已经有 XX 年工作经验了。

（紧接着， 聊 4+1）

(1)在 **Linux 系统运维**方面，我熟悉 Linux 的日常运维工作，能够编写 Shell 脚本，提升自动化运维的效率；

(2)在**数据库**方面，我熟悉 SQL 的增删改查等常用命令的操作；熟悉 MySQL 数据库主从架构的搭建，熟悉备份和还原等运维操作；

(3)在**中间件**方面，我熟悉 Nginx、Tomcat 能组件的使用和基本配置；

(4)在**云计算**方面，我熟悉 Docker、K8s 等技术栈，也熟悉业界阿里云等云产品。

（加 1）在**开源组件**方面，**自动化运维领域，**我熟悉 Ansible 自动化运维组件的使用；**云下监控方面，**我熟悉 Zabix 组件的基本配置和使用；**云上监控方面，**我熟悉 Prometheus 云上监控组件的配置和使用；**DevOps 方面，**我熟悉 CI/CD 持续集成和交付方面的理论知识和操作步骤，熟悉 Git、Jenkins 等组件的配置和使用。

（比如，CI 流程 5 大阶段：提[代码]、拉[代码]、编[译代码]、测[单元测试/集成测试]、查[质量检查]）

（注意，三年及以下运维经验的，在开源组件上挑选最多两个就行，不需要全部都写，可以思考一下——企业里有分工协作，不会什么都干；面试时也容易被挑战）

2.2 个人经历

本科毕业于 XXX 学校，XXX 专业（如果是计算机相关专业，可以扩充：在校期间系统学习了计算机网络、操作系统、数据库等核心课程）。毕业以后，我先后在 XXX 公司和 XXX 公司从事运维工作，积累了丰富的实战经验。

在最近的一家公司，我担任运维工程师一职，主要负责 XXX 模块的业务，协助完成了 XXX 项目的运维工作，提高了 XXX 的运维效率，完成了部门交代的相关任

务。

2.3 项目介绍

（比如，主要参与了 K8s 三主多从集群架构的搭建和微服务部署，基于 Kubernetes v1.24 版本，使用 kubeadm 部署高可用集群（3 Master + 5 Worker），并集成 Calico 实现网络策略，通过 Helm Chart 管理微服务部署，能够熟练完成部门交代的任务，提高运维工作效率）

（比如，主要参与了 MySQL 数据库主从架构的部署，搭建 MySQL Group Replication 集群，配置读写分离，支撑了业务侧的访问请求，通过慢查询日志分析，协助开发同事降低了查询耗时，提升了页面访问体验。）

（比如，主要参与了公司核心业务系统的容器化迁移工作。协助完成了从传统虚拟机（VM）到 Kubernetes 的迁移，完成 20+微服务的容器化改造。）

2.4 离职原因

在上一家公司，我积累了扎实的[如系统运维/K8s 集群管理]经验，由于[公司业务收缩、团队重组或技术路线变更]，我希望聚焦个人职业规划和长远发展，在外面寻找机会。

注意，不要超过 30 秒。

2.4.1 话术避雷

在描述离职原因时，注意一些话术。

- 强调**“希望聚焦技术深度”“业务方向调整”**；不说“领导能力差”“团队内耗”“工资低”；
- 表达**“能力强”**；不说“原公司不足”；
- 表达**“能力匹配新公司”**，不说“我想来学习学习”；

第三章 公司话术篇

要确保下面的问题，都能回答上来。换句话说，围绕下面这些问题，丰富公司话术。

3.1 关于人员规模

3.1.1 你们公司规模多大

参考**【经验分享】**部分的**【运维人员规模】**。

3.1.2 你们运维部门多少人，怎么分工的

一个人直接管理的人数不超过 10 人；超过之后，会分组，安排运维小组长；比如，天津某家公司，40 个研发人员的团队，其中运维 14 人，4 个硬件、4 个硬件+网络、4 个软件；

小型公司（<100 人）：5－10 名运维人员；

中型公司（100~500）：10－30 名运维人员；

大型公司（>500 人）：超过 50 名运维人员；

3.1.3 你们开发部门多少人

一般是二八配比，即

开发:运维 = 8:2

3.2 关于服务器规模

3.2.1 你们运维维护的服务器规模是多少

根据公司业务规模进行具体分析。

小型公司（<100 人）：上云；

中型公司（100~500）：上云、或者 10-50 台服务器，每台 2-3w 左右；

大型公司（>500 人）：上云，或者 50-2000 台服务器，更大的公司具体分析；

3.2.2 你自己负责多少服务器

应届实习，一般在 10 台以内；

毕业三年以内，一般在 10-30 台；

毕业三五年以上，根据自己能力，一般都是多人合作，大概管理 100-200 台；

3.2.3 你们的服务器型号

业界的大部分公司，采购的云下主机，**一般单台在 2-3.5w 之间居多**。单台服务器的典型配置如下：

- 型号：浪潮 NF5280M6
- CPU：1 × 英特尔至强银牌 4310（12 核/24 线程）
- 内存：64GB DDR4 ECC
- 存储：2 × 480GB SSD + 4 × 2TB HDD
- 价格：约 1.9-2.2 万元（含基础服务）

常见服务器的图片如下：

服务器型号一般是**浪潮（如 NF 系列或 SA 系列）、Dell、联想、华为（泰山、鲲鹏、超聚变）**居多。整体来讲，华为的服务器，相对贵一些。

3.2.4 你这个项目用了多少服务器

根据项目规模而定。一般一台服务器，会拆分出 4-6 台虚拟机。单台虚拟机大部分以下两种配置:

单台虚拟机配置		
配置	**标准版**	**高性能版**
CPU	8 核 16 线程	16 核 32 线程
内存	32G	64G
硬盘	300G	512G

关于 CPU 配置，补充信息如下

配置类型	CPU 型号示例	物理核数	适用场景
单路入门级	英特尔至强银牌 4310	12 核 24 线程	轻量级虚拟化/数据库
单路主流型	英特尔至强金牌 6330N	28 核 56 线程	云计算节点/中型应用
双路性价比型	2×AMD EPYC 7302P	16 核×2=32 核 32 线程×2=64	高密度计算/存储服务器
双路高性能型	2×英特尔至强金牌 5418Y	24 核×2=48 核	大数据分析/ERP 系统

3.3 关于发展

3.3.1 关于下一份工作的期待，包括工作内容和创造的价值

下面围绕上面问题，给出一些常见业务领域的公司话术。

3.4 面向政府行业的软件销售公司

3.4.1 公司业务

公司是一个 to G（面向政府）的业务公司，卖给企业和政府相应的办公软件和管理软件。

3.4.2 公司规模

公司规模 500-1000 人。

3.4.3 公司资产

200 台服务器，服务器型号是**浪潮 XXX**；每套开发测试环境需要 10 台服务器，搞了 4 套环境。除此之外，就是其他办公系统的环境。公司内部暂时没有云上的服务器。

3.4.4 软件规模

一套系统下来，200w 的项目，大概 10 台主机，搭建一个 K8s 集群，三主七从。其中中间件三台，APP 应用 4 台。

3.4.5 开发团队

开发大概 80 人，1 个架构师，5 个小组长，负责 10 个模块，都是跟业务系统相关的。

3.4.6 运维团队

运维团队 35 人。其中 20 人左右是驻场，不怎么在公司，15 个人在公司，其中，5 个人系统和 K8s 运维,5 个数据库中间件的,3 个负责不断推倒和重建开发测试环境，1 个手册，1 个组长。

3.4.7 我负责的内容

个人带入，可以选择系统和 K8s 运维、数据库运维、环境部署、外出实施几个混搭着搞，也可以三年干过两个职责。

对应上自己的个人职责。比如，

环境部署，就是擅长 K8s 集群的搭建和初始化配置，一个人负责三个环境，每个 10 台服务器。

K8s 运维，就要回答生产上印象深刻的问题。

外出实施，要回答跟客户沟通中印象深刻的问题。

第四章 项目话术篇

每个项目都不一样，问的方向也不一样，这里我们五五开，提供一些公共问题，后面再针对不同项目，提供剩下 50%的问答。

注意事项

第一，一个人不能面面俱到，业务必须聚焦，选择其中三个左右，主攻下来。下面给出一些特定项目的常见话术。

有的人会问，项目三四个，会不会被质疑水平太差？我们和 Java 开发不一样，开发上四个月一个项目，属于正常。运维上偏持续运维，一般一两年一个，属于正常。

第二，因为你简历上项目写什么技能，就会被问到什么。结合简历，打造一个适合自己难度的话术文档。个人技能，可以适当丰富很多。

有的人会问，技能太少，会不会到时候 HR 都不看。个人技能可以多写一些，了解的也写上去，方便 HR 命中技能。

有的人会问，项目技能太少，终面会不会压低工资。目前市场上给定薪资，主要看学历、年龄、工作年限、上一份工作的公司规模和项目技术栈。项目技术栈一般是固定的，能答上来越多，给的工资会浮动 2K 左右。可以写少答多，给出惊喜；所以还得在后面话术上，问到的，80%以上都可以答上来，就基本能过了。

第三，对于选定的项目，老师提到的【面试题】，务必掌握。

4.1 Shell 和 Linux 运维

4.1.1 面试必问

4.1.1.1 说出你常用的 10 个 Linux 命令

开放题目。面试官可以根据答案大概知道操作水平和日常工作方向。

- 日志处理：三剑客（grep、sed、awk）；
- 性能监控：df、top、du、free；
- 服务、权限管理：systemctl、ps、chown、chmod；
- 网络管理：ss、netstat；
- 配置管理：mkdir、touch、vim、source；
- 其他日常命令：cd、ls、mkdir、echo；

4.1.1.2 命令 df 和 du 的区别是什么

特性	df（disk free）	du（disk usage）
统计对象	文件系统（Filesystem）的整体空间	目录或文件的实际占用空间
数据来源	读取文件系统元数据	遍历目录计算文件大小
显示单位	默认以块（block）为单位	默认以字节为单位
耗时情况	几乎瞬时完成	耗时随目录深度和文件数量增加

4.1.1.3 命令 grep 中，-A、-B、-C、-E、-P 表示什么意思

自己可以使用 grep --help 进行查看。

参数	作用	典型场景
-A	显示匹配行及后 N 行	查看错误后的恢复操作
-B	显示匹配行及前 N 行	查看错误前的配置或操作
-C	显示匹配行及前后各 N 行	获取完整的错误上下文
-E	使用扩展正则表达式语法	编写更复杂的匹配模式（如或、量词）

4.1.1.4 Linux 种 755 644 区别

755 权限是 rwxr-xr-x（文件可执行），644 是 rw-r--r--（文件不可执行）。

4.1.1.5 命令 vim 中，怎么删除当前行和下面的 9 行

4.1.1.6 我见你写的会 Shell 自动运维，一般用来干哪些事情

可以随机选择三个，来发挥：

1、系统基础管理

- 文件操作：自动清理日志、备份重要文件（如/etc 目录）、监控磁盘空间（df -h | grep -E '/$'）
- 服务管理：批量重启服务（如 systemctl restart nginx）、检查服务状态（systemctl is-active）
- 用户管理：批量创建用户、修改权限（如 chmod 755 /scripts）

2、批量处理任务

- 批量文件处理：重命名、压缩、移动文件（如 find /data -name "*.log" -exec gzip {} \;）
- 远程执行命令：通过 SSH 批量管理服务器（如 for host in $(cat hosts.txt); do ssh $host 'uptime'; done）
- 数据库操作：定时备份 MySQL（mysqldump）、清理过期数据

3、应急响应与故障恢复

- 自动修复：检测到服务崩溃时自动重启（如 ps -ef | grep nginx || systemctl start nginx）
- 灾备切换：当主服务器故障时，自动切换到备机
- 安全审计：定期检查系统日志，识别异常登录（如 lastb | awk '{print $3}' | sort | uniq -c）

4、配置管理

- 标准化环境：通过脚本统一配置多台服务器（如 Nginx、MySQL 参数）
- 合规检查：检查服务器是否符合安全基线（如 SSH 端口、防火墙状态）

4.1.1.7 怎么删除 7 天前的日志

考察 find 命令的-mtime 参数，+7 表示 7 天前。使用 find 查找文件，-mtime 按照修改时间找到 7 天前的，-delete 删除掉即可。

```
#!/bin/bash
find /var/log/tomcat -name "*.log" -mtime +7 -exec rm {} \;

或者
find $LOG_DIR -type f -mtime +7 -delete
```

4.1.1.8 怎么做系统运维中的服务器监控

```
#!/bin/bash
if ! pgrep -x "tomcat" > /dev/null; then
    systemctl restart tomcat
fi
```

4.1.1.9 怎么查看某一个进程的 CPU 使用率，怎么进一步查看里面线程的使用率

top 命令，在展示的视图里，按 H 表示进入线程查看模式；按 P 表示按照 CPU 使用率降序排列；按 M 表示按照内存降序排列。

4.1.1.10 怎么查看内核

命令 uname -a 或 uname -r。

4.1.1.11 关于 swap 分区的作用

swap 分区就好比是电脑的“备用内存仓库”。

当电脑里正在运行的程序太多，把正常的内存（就像一个小仓库）都占满了，没有地方放新的数据了，这时候系统就会把一些暂时不用的数据先放到 swap 分区（一个在硬盘上专门划分出来的空间，就像一个大的备用仓库）里，这样正常的内存就有空间去处理其他更紧急的任务了，让电脑能继续运行更多程序，不至于因为没地方存数据而死机。

硬盘读取速度比较卡，一般不能完全依赖 swap。

4.1.1.12 Shell 中，$@、$#、$*、$? 之类的是什么意思

位置参数变量

变量	含义
$0	当前脚本的文件名（如 ./script.sh）。
$1~$9	脚本的第 1 到第 9 个参数。超过 9 个参数需用${10}、${11}等。
$@	所有参数的列表，每个参数保持独立（如 "$1" "$2" ...）。常用于循

变量	含义
	环遍历参数。
$*	所有参数的列表，合并为一个字符串（如"$1 $2 ..."）。
$#	参数的数量（不包括脚本名$0）。例如，执行 script.sh a b c 时，$# 为 3。

状态与进程变量

变量	含义
$?	**上一个命令的退出状态码**: - 0 表示成功 - 非零值（如 1、127）表示失败。常用于检查命令是否执行成功。
$$	当前脚本的进程 ID（PID）。例如，可用于创建临时文件: temp_file="/tmp/myscript_$$.log"。
$!	最后一个后台进程的 PID。例如，sleep 10 & echo "后台进程 PID: $!"。
$-	当前 Shell 的选项标志（如 h、x 等），用于显示当前启用的 Shell 选项。

4.1.1.13 怎么实时监控日志文件的变化

tail 命令。

4.1.1.14 怎么快速清空一个大文件，注意，不是删除它

考查 > 重定向符的使用。

4.1.1.15 解释一下 grep -v "error" log.txt | awk '{print $2}' | sort | uniq -c 的作用

4.1.1.16 如何用 Shell 统计一个文件中每个单词的出现频率

4.1.1.17 如何用 sed 或 awk 提取 nginx.conf 中所有 server_name 后的域名

4.1.1.18 命令 less 和 more 的区别

4.1.1.19 命令 man 的作用

4.1.1.20 符号>和>>的区别

4.1.2 项目常问

4.1.2.1 从 0 到 1 完成运维搭建工作后，需要输出哪些文档

输出一系列详细且规范的文档，这有助于后续的维护、管理、问题排查以及知识传承。以下是可能需要输出的文档类型：

项目概述类

项目实施计划文档

- 明确项目的各个阶段和里程碑，规划每个阶段的开始时间、结束时间和主要任务。
- 制定资源分配计划，涵盖人力、物力、财力等方面的资源使用情况。

系统配置类

服务器配置文档

- 记录服务器的硬件信息，如 CPU、内存、硬盘、网卡等的型号和参数。
- 详细说明服务器的操作系统安装过程、系统配置参数，包括网络配置、用户权限设置、防火墙规则等。

软件系统配置文档

- 针对搭建的各类软件系统，如数据库、中间件、应用程序等，记录其安装步骤、配置参数和依赖关系。
- 说明软件系统的初始化设置和数据导入过程。

网络配置文档

- 绘制网络拓扑图，标注网络设备的型号、位置和连接方式。
- 记录网络设备的配置信息，如交换机的 VLAN 划分、路由器的路由策略、防火墙的访问控制规则等。

操作维护类

日常操作手册

- 提供系统的日常操作指南，包括启动、停止、重启服务的命令和步骤。
- 说明数据备份和恢复的操作流程、定时任务的设置方法等。

故障处理手册

- 列举常见故障的现象、原因和解决方法，建立故障处理的流程和应急预案。
- 提供故障排查的工具和方法，以及与供应商的联系信息。

4.1.2.2 CPU 负载过高，如何快速定位问题

当 CPU 负载过高时，可按以下步骤快速定位问题：

1、确认高负载情况。通过 top 命令查看系统整体的 CPU 使用情况。

2、找出高 CPU 占用进程。在 top 或 htop 中，按 CPU 使用率排序（在 top 中按 P 键）

3、分析高 CPU 进程。对于 Java 等多线程应用，使用 top -Hp <PID>查看该进程下各个线程的 CPU 占用情况，找出占用 CPU 高的线程。

4.1.2.3 慢查询 SQL 问题排查与解决

在我负责的一个电商项目中，随着业务的发展和用户量的增长，数据库查询性能逐渐下降，特别是在高峰期，部分关键业务功能的响应时间明显延长，严重影响了用户体验。通过监控和日志分析，我们发现某些 SQL 查询执行特别慢，成为了性能瓶颈。这些慢 SQL 查询主要涉及复杂的关联查询、大数据量的筛选以及缺乏有效索引的字段。由于数据库表结构设计不合理、索引使用不当以及查询逻辑复杂等原因，导致查询效率低下，CPU 和 IO 资源消耗过高。

问题解决过程：

为了解决这些慢 SQL 问题，我们采取了一系列调优措施：首先，我们对慢 SQL 查询进行了定位和分析。利用数据库的性能监控工具，我们找出了执行时间最长、资源消耗最多的 SQL 语句。然后，对这些 SQL 语句进行了详细的解释分析，查看了查询计划、扫描行数、索引使用情况等信息，确定了性能瓶颈所在。接下来，我们针对每个慢 SQL 查询进行了优化。对于复杂的关联查询，我们尝试简化查询逻辑，减少不必要的关联操作；对于大数据量的筛选，我们优化了筛选条件，减少了返回结果集的大小；对于缺乏有效索引的字段，我们添加了合适的索引，提高了查询速度。在优化过程中，我们还对数据库表结构进行了调整。根据业务需求和查询特点，我们重新设计了部分表结构，提高了数据的存储和查询效率。同时，我们也对索引进行了重建和维护，确保索引的有效性和准确性。除了上述措施外，我们还对数据库进行了整体性能调优。包括调整数据库参数、优化缓存策略、增加硬件资源等，以提高数据库的整体性能。

最终落地的方案：

经过一系列的调优措施，我们成功解决了慢 SQL 查询问题，提高了数据库查询性能。具体的落地方案包括：

- 优化了多个关键业务的 SQL 查询逻辑，减少了不必要的关联和筛选操作；
- 添加了多个缺失的索引，提高了查询速度和效率；
- 调整了部分表结构，使其更符合业务需求和查询特点；
- 对数据库进行了整体性能调优，提高了稳定性和响应速度。

在实施这些方案后，我们进行了性能测试和验证。通过对比优化前后的数据，我们发现关键业务的响应时间明显缩短，数据库资源消耗也大大降低。同时，用户体验也得到了显著提升，满意度明显提高。这次慢 SQL 调优的经历让我深刻体会到了性能调优在项目开发中的重要性。在未来的工作中，我将更加注重代码质量和数据库性能的优化，努力提升系统的性能和用户体验。

4.1.2.4 存储服务器文件上传卡死问题排查与解决

在公司的业务架构中，有一台专门用于存储文件资料的存储设备服务器。随着业务的不断发展，文件上传的需求日益增长，但这台服务器频繁出现上传卡死的现象，严重影响了业务流程的顺畅进行，急需进行深入排查并彻底解决该问题。

初步排查：

登录操作系统检查系统资源与进程情况

我第一时间登录到上传服务器的操作系统，借助 top、free、iostat 等命令，查看 CPU 使用率、内存剩余空间、磁盘 I/O 读写速度等关键资源指标，确认是否存在因资源耗尽（如 CPU 长时间满载、内存不足导致频繁交换、磁盘 I/O 阻塞）而引发的卡死。从监控数据来看，在上传卡死的时间段，CPU 使用率偶尔会飙升至 90% 以上，但内存和磁盘 I/O 似乎并无明显异常。

同时，通过 ps aux 命令查看系统进程，检查是否有过多的上传进程被创建，或者是否存在僵尸进程占用系统资源，发现进程数量处于正常范围，未出现进程满载或僵尸进程泛滥的情况。

检查磁盘识别状态

在操作系统中，我使用 fdisk -l 等命令查看是否能够正常识别到插入的本地磁盘设备，经检查，所有本地磁盘均被正确识别且挂载状态正常，初步排除本地磁盘故障导致上传卡死的可能性。

发现问题阶段：

使用 df 命令初步定位问题范围

当服务器再次出现卡死现象时，我立即使用 df -h 命令查看本地磁盘空间使用情况，命令能够正常执行，显示各本地挂载点的磁盘使用率均在合理范围内。然而，当我尝试使用 df -h 命令查看网络存储设备的挂载点时，命令直接卡死，毫无响应。基于这一现象，我初步判断问题很可能与网络存储设备的挂载或网络连接有关，因为网络设备的异常很可能会阻塞文件向远程存储的上传流程。

继续深入挖掘：

查看系统日志挖掘更多线索

我迅速查看系统的日志文件，重点查看 /var/log/messages、/var/log/syslog 等系统日志以及应用程序自身的日志。在系统日志中，发现在上传卡死的时间段，有较多与网络文件系统（如 NFS）相关的报错信息，例如 “NFS: server not

responding”“NFS: retrans timeout” 等，这进一步印证了网络存储设备连接可能出现问题的猜想，同时也让我意识到可能是网络层面的故障导致了文件上传卡死。

联系开发团队协查应用层问题

我及时联系了开发团队，共同梳理近期是否有涉及文件上传模块的代码改动，以及应用层是否存在其他潜在的报错情况。开发人员反馈，在最近一次代码更新中，对文件上传的超时时间和重试机制进行了调整，但这看似合理的调整可能并未充分考虑到网络异常情况的复杂性，导致在面对网络存储设备偶尔的延迟或无响应时，应用无法有效处理，进而出现卡死现象。同时，应用日志中也记录了一些与网络连接超时相关的错误，如 “Connection timed out while uploading file” 等。

由于问题已初步指向网络存储设备，我立即联系存储团队。存储团队对网络存储设备进行了全面检查，包括存储设备的网络连接状态、存储池的空间使用情况、网络带宽是否充足、存储设备的配置参数等。经过排查，发现网络存储设备在特定高负载情况下，会出现网络连接抖动和响应延迟的问题，而且存储设备的网络配置参数与服务器端的设置存在一定的不兼容情况，导致文件上传过程中容易因网络问题而卡死。针对这些问题，存储团队采取了以下解决措施：一是优化网络存储设备的网络配置，调整了网络超时时间和重传参数，使其与服务器的网络环境更加匹配；二是对存储设备的网络连接进行了优化和加固，修复了一些潜在的网络连接故障点，提升了网络连接的稳定性；三是增加了网络带宽的监控和管理，确保在网络高负载情况下仍能保障文件上传的顺畅进行。

在存储团队完成网络存储设备的优化和调整后，我与开发团队再次进行了联合测试。在相同的业务场景下，模拟大量文件的上传操作，观察服务器的运行状态和文件上传进度。经过多次测试，发现服务器的上传卡死问题得到了有效解决，文件能够稳定、顺利地上传至存储设备，系统日志和应用日志中也未再出现与之前类似的网络连接错误和卡死报错信息。

经验教训总结

通过这次服务器上传卡死问题的排查与解决，我深刻认识到：

在面对复杂的运维故障时，需要从系统资源、网络连接、应用代码、存储设备

等多方面进行全面排查，逐步缩小问题范围，不能仅仅局限于某一个层面。

系统日志和应用日志是排查故障的重要依据，及时、准确地查看和分析日志信息能够为我们提供关键的线索，快速定位问题根源。

不同团队之间的紧密协作至关重要。运维团队与开发团队、存储团队的高效沟通与协同工作，能够充分发挥各方的专业优势，加快问题解决的进度。

此次事件也暴露了我们在代码更新测试和网络存储设备管理方面的不足。在后续的工作中，我们需要进一步完善代码更新的测试流程，充分考虑各种可能的异常场景，并加强对网络存储设备的监控和维护，定期进行巡检和优化，确保业务的稳定运行。

4.1.3 个人职责

模板 1：基础运维与自动化

- 负责 Linux 服务器的日常运维管理，包括系统安装、用户权限管理和服务配置；
- 编写 Shell 脚本实现日常运维工作的自动化，如日志处理、文件备份等；
- 使用 Ansible 进行服务器批量配置管理和应用部署；
- 维护系统运行环境，处理软件包依赖和版本管理；
- 协助处理系统故障，执行基本的故障排查和恢复；

模板 2：云平台运维

- 负责云服务器(如 AWS/阿里云)的日常运维管理工作；
- 使用 Terraform 进行云资源的管理和配置；
- 实现云主机的自动化初始配置和环境准备；
- 维护云上数据库、缓存等中间件服务；
- 参与云迁移项目的实施和支持；

模板 3：监控与日志管理

- 部署和维护监控系统，配置监控项和告警规则；
- 编写日志分析脚本，定期生成系统运行报告；
- 搭建和维护日志收集分析平台；
- 分析系统性能数据，发现潜在问题；
- 参与生产环境故障的排查和处理；

模板 4：DevOps 支持

- 维护 CI/CD 流水线，支持应用的自动化构建和部署；
- 管理代码仓库和制品库，支持版本发布流程；
- 使用配置管理工具实现基础设施即代码；
- 协助容器化应用的部署和维护；
- 优化持续集成和交付流程；

模板 5：系统安全与维护

- 执行系统安全加固，包括权限管理和漏洞修复；
- 维护防火墙规则和网络安全配置；
- 实施系统备份策略，确保数据安全；
- 监控系统安全状态，处理安全事件；
- 参与安全合规检查和安全加固工作；

4.2 中间件 Nginx

4.2.1 面试必问

4.2.1.1 Nginx 的默认轮询三种策略

1、IP 哈希

2、RR 算法；

3、加权 RR 算法；

4.2.1.2 Nginx 的 alias 和 root 有什么区别

root：root 指令会把请求的 URI 直接拼接到 root 指定的路径后面，形成完整的文件查找路径。

alias：alias 指令会将 location 匹配的部分替换为 alias 指定的路径，从而得到文件的查找路径。

4.2.1.3 Nginx 在运维方面的调优有哪些？默认 Nginx 可以支撑多大的访问量

说出两个就可以。

1、连接处理优化 worker_processes、worker_connections；

2、静态资源的缓存优化；

3、安全优化 limit_req 和 limit_conn；

4.2.1.4 Nginx 如何配置实现负载均衡

Nginx 配置负载均衡步骤如下：

1. 安装 Nginx。
2. 在配置文件中用 `upstream` 定义后端服务器组，列出各服务器地址和端口。
3. 在 `server` 块的 `location` 中，使用 `proxy_pass` 指令将请求转发到后端服务器组。
4. 根据需要选择轮询、加权轮询、IP 哈希、最少连接等负载均衡算法。
5. 修改配置后，用 `nginx -t` 检查语法，`nginx -s reload` 重载配置使生效。

4.2.2 项目常问

4.2.3 个人职责

模板 1：系统运维方向

- 负责 Nginx 服务器的日常监控、维护和故障排查，保障服务稳定可用。
- 处理用户提交的 Nginx 访问异常工单，解决域名解析、页面加载失败等问题。
- 协助后端工程师配置 Nginx 与应用服务器的反向代理和负载均衡。
- 参与服务器资源管理，定期检查 Nginx 的 CPU、内存和连接数使用情况。
- 编写 Shell 脚本自动化 Nginx 日志清理、配置检查等日常运维任务。

模板 2：系统运维方向|系统升级

- 协助搭建 Nginx 测试环境，部署 SSL 证书和访问控制策略。
- 监控 Nginx 版本升级过程中的服务状态，记录 502、504 等错误日志并反馈。
- 参与 Nginx 配置迁移工作，验证新配置的功能和性能表现。
- 配合测试团队完成 Nginx 功能和负载测试，确保升级后稳定运行。
- 整理 Nginx 部署、升级文档及常见问题解决方案。

模板 3：云服务与监控方向

- 负责云上 Nginx 实例管理，配置安全组规则和网络访问策略。
- 使用 Prometheus 和 Grafana 搭建 Nginx 监控平台，跟踪 QPS、响应时间等指标。

- 处理 Nginx 资源告警，分析流量突增、连接超时等异常情况。
- 协助制定 Nginx 日志备份策略和灾备演练计划。
- 参与 Nginx 自动化运维工具的研究和测试，提升部署效率。

模板 4：故障排查与用户支持方向

- 负责 Nginx 虚拟主机管理，配置域名绑定和权限控制。
- 处理用户反馈的 Nginx 访问慢、页面无法显示等问题。
- 分析 Nginx 访问和错误日志，定位故障原因并协同解决。
- 参与 Nginx 服务器和关联设备的日常巡检，记录运行状态。
- 协助编写 Nginx 运维手册和常见问题解决方案，供团队参考。

模板 5：故障排查与用户支持方向

- 参与 Nginx 高并发压测，记录流量峰值时的性能表现。
- 配合团队优化 Nginx 配置参数，如缓存策略、连接池大小。
- 整理 Nginx 性能优化过程中的技术文档和测试报告。
- 协助测试团队验证优化后 Nginx 的稳定性和吞吐量。

4.3 Docker 基础

4.3.1 面试必问

4.3.1.1 你自己写过 Dockerfile 吗？怎么写的

4.3.1.2 其中 copy 和 add 的区别

4.3.1.3 Docker 的几种网络模式

Host、Bridge、None、Overlay；

4.3.1.4 Docker 容器之间怎么通信的

在 Docker 环境中，容器间通信的方式受其网络模式影响，下面为你详细介绍不同网络模式下容器的通信机制：

Bridge 模式是 Docker 的默认网络模式。在此模式里，Docker 守护进程会创建名为 docker0 的虚拟网桥，容器借助 veth 设备对连接到该网桥上。

1、同一宿主机内的容器通信：每个容器都会被分配一个独立的 IP 地址，同一宿主机内的容器能通过 IP 地址直接互相访问。例如，若容器 A 的 IP 地址是 172.17.0.2，容器 B 的 IP 地址是 172.17.0.3，那么容器 A 可通过 172.17.0.3 访问容器 B 开放的端口。

2、不同宿主机间的容器通信：若要实现不同宿主机上的容器通信，需进行端口映射，把容器端口映射到宿主机的端口上，然后通过宿主机的 IP 地址和映射的端口进行访问。

4.3.2 项目常问

4.3.3 个人职责

模板 1：容器化部署与维护

- 负责应用容器化改造，编写和维护 Dockerfile
- 管理 Docker 镜像仓库，维护镜像版本和依赖关系
- 部署和维护容器化应用运行环境
- 监控容器运行状态，处理容器异常问题
- 优化容器资源使用，调整容器配置参数

模板 2：Kubernetes 集群运维

- 协助部署和维护 Kubernetes 生产集群
- 使用 Helm 管理应用部署和配置
- 监控 Kubernetes 集群和 Pod 运行状态
- 处理容器编排相关的故障和性能问题
- 维护 Ingress 控制器和服务发现配置

模板 3：CI/CD 与容器集成

- 将 Docker 集成到 CI/CD 流程中，实现自动化构建
- 维护基于容器的持续集成环境

- 优化镜像构建流程，减少构建时间和镜像体积
- 管理多阶段构建和镜像分层
- 配合开发团队优化容器化部署方案

模板 4：容器安全与优化

- 实施容器安全最佳实践，包括镜像扫描
- 配置容器运行时安全策略
- 优化容器资源限制和调度策略
- 维护容器网络和存储配置
- 处理容器相关的安全漏洞和合规要求

模板 5：微服务架构支持

- 支持微服务容器化部署和运维
- 维护服务网格(Service Mesh)基础设施
- 配置和管理 API 网关与容器化服务集成
- 监控微服务链路和容器性能指标
- 参与微服务架构的容量规划和扩展

4.4 K8s

4.4.1 面试必问

4.4.1.1 能说出 K8S 的两大类型节点和七大组件

Master: etcd、API-Server、Scheduler、Controller-Manager

Node： kubelet、Kube-proxy、runtime

4.4.1.2 能说出四种控制器，掌握其中的 Deployment、DaemonSet 两种

Deployment、DaemonSet、StatefulSet、Job、CronJob

4.4.1.3 能说出 Pod 的创建三大步骤和销毁的三大步骤

创建

1、向 API Server 发申请，写到 etcd

2、scheduler 分配节点，写道 etcd

3、kubelet 认领任务，下发创建任务

4、Runtime 具体创建

销毁

1、向 API Server 发申请

2、Kubernetes 把 Pod 从节点中移除，停止接受请求；

3、Pod 完成停止前的任务

4、容器停止，Pod 资源释放

4.4.1.4 能说出四种 Service，掌握其中的 ClusterIP、NodePort 两种

ClusterIP、NodePort、LoadBalancer、ExternalName

4.4.1.5 能说出 5 种存储卷 Volume，知道从哪里查看版本支持的存储卷

映射：EmptyDir、HostPath、Network File System；

etcd：ConfigMap、Secret

4.4.1.6 Ingress 是 K8S 当中的 Nginx

K8s、Nginx、Traefik

4.4.1.7 说出在 K8s 集群中，从前端页面到访问具体 Pod 的链路吧

链路：用户浏览器 -> DNS 解析 -> 外部负载均衡器(可选) -> Ingress Controller -> Service -> Pod。处理完，原路返回。

其中，外部负载均衡器（可选）。在生产环境中，通常会使用外部负载均衡器（如云提供商的负载均衡服务，像 AWS ELB、阿里云 SLB 等）来接收外部流量。负载均衡器会根据一定的算法（如轮询、最少连接等）将请求分发到 K8s 集群中的节点。就是我们上课学的，Load Balancer。

4.4.1.8 说出服务需要在 K8s 上运行的步骤，介绍一下 Yaml 的一些配置参数

将服务部署到 Kubernetes (K8s) 上运行，通常需要经历容器化应用、创建 K8s 资源清单、部署到 K8s 集群等步骤。

1、容器化应用

a. 编写 Dockerfile，比如

```
# 使用 Python 基础镜像
FROM python:3.9-slim


# 设置工作目录
```

```
WORKDIR /app

# 复制依赖文件
COPY requirements.txt .

# 安装依赖
RUN pip install --no-cache-dir -r requirements.txt

# 复制应用代码
COPY . .

# 暴露应用端口
EXPOSE 5000

# 启动应用
CMD ["python", "app.py"]
```

b. 构建容器镜像

c. 推送镜像到仓库

2、创建 K8s 资源清单

Deployment 资源

Service 资源

3、部署到 K8s 集群

创建 Deployment 和 Service

验证部署

4、访问服务，进行验证

4.4.1.9 关于 Helm Chart 的相关工作

在服务发布的时候，制作了一个 Helm Chart。文件整体三部分：Chart.yaml、templates、values。

```
mychart/
├── charts/
├── Chart.yaml
```

```
├── templates/
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   └── service.yaml
└── values.yaml
```

4.4.1.10 Kubernetes（K8s）中通常会用到以下几种证书

CA 证书：即根证书，是整个证书体系的信任根。

Master 上：API Server 证书、etcd 证书

Node 上：kubelet 证书、ServiceAccount 证书

4.4.1.11 Scheduler 和 Controller-Manager 怎么交互和通信的

在 Kubernetes 中，Scheduler 与 Controller－Manager 通过 API Server 进行交互通信。

1、Scheduler 负责监控新创建且未分配节点的 Pod，根据调度算法为其挑选合适节点 。

2、当 Scheduler 完成 Pod 调度后，会通过 API Server 更新 Pod 的节点分配信息，

3、Controller－Manager 通过监听 API Server 上 Pod 资源的变化，获取调度结果并进行后续处理，比如确保对应节点启动 Pod 等，

它们借助 API Server 实现松耦合，协同保障集群稳定运行。

4.4.1.12 Deployment、StatefulSet、DaemonSet 的区别

Deployment：广泛应用于各种无状态应用的部署和更新。例如，部署 Web 应用、微服务等。

StatefulSet：为每个 Pod 分配一个唯一的标识符和稳定的网络域名，并且可以挂载独立的存储卷。

适用于需要保持状态的应用，如数据库、消息队列等。比如部署 MySQL 集群、Zookeeper 集群等。

DaemonSet：部署一些需要在每个节点上都运行的组件或工具，如日志收集工具

filebeat。

4.4.1.13 Nginx 的默认轮询三种策略

1、IP 哈希

2、RR 算法；

3、加权 RR 算法；

4.4.1.14 Nginx 的 alias 和 root 有什么区别

root：root 指令会把请求的 URI 直接拼接到 root 指定的路径后面，形成完整的文件查找路径。

alias：alias 指令会将 location 匹配的部分替换为 alias 指定的路径，从而得到文件的查找路径。

4.4.1.15 Nginx 在运维方面的调优有哪些？默认 Nginx 可以支撑多大的访问量

说出两个就可以。

1、连接处理优化 worker_processes、worker_connections；

2、静态资源的缓存优化；

3、安全优化 limit_req 和 limit_conn；

4.4.1.16 Nginx 如何配置实现负载均衡

Nginx 配置负载均衡步骤如下：

5. 安装 Nginx。

6. 在配置文件中用 `upstream` 定义后端服务器组，列出各服务器地址和端口。

7. 在 `server` 块的 `location` 中，使用 `proxy_pass` 指令将请求转发到后端服务器组。

8. 根据需要选择轮询、加权轮询、IP 哈希、最少连接等负载均衡算法。

5. 修改配置后，用 `nginx -t` 检查语法，`nginx -s reload` 重载配置使生效。

在 Kubernetes 里，Deployment 一般不用于部署数据库，主要有以下几个原因：

1. 数据库状态管理复杂

数据库是有状态的应用，需要持久化存储数据。而 Deployment 主要面向无状态应用设计，默认滚动更新时 Pod 会被销毁重建，新 Pod 无法保证能获取到之前的数据，不利于数据的持久保存和管理。

2. 难以保证数据一致性

Deployment 的滚动更新机制在更新过程中会逐步替换旧 Pod 为新 Pod。但数据库在更新时，需要保证数据在新旧版本间的一致性，Deployment 简单的滚动更新难以确保在这个过程中数据不会出现不一致问题，比如数据丢失、数据版本不兼容等。

3. 缺乏精细的伸缩控制

数据库的伸缩需要根据实际的负载、数据量等因素进行精细控制，比如读写分离、水平和垂直扩展等。Deployment 的伸缩策略相对简单，主要基于副本数量的调整，不能很好地满足数据库复杂的伸缩需求。

4.4.1.17 怎么使用 Deployment 实现 DaemonSet 的要求

DaemonSet 会在每个符合条件的节点上都运行一个 Pod 实例，而 Deployment 默认不会这样做。不过，你可以通过特定的配置，使用 Deployment 来近似实现 DaemonSet 的要求。

利用节点标签和 Pod 调度规则，让 Deployment 创建的 Pod 能在每个目标节点上运行。比如使用节点亲和性或节点标签，就可以尽可能做到 DaemonSet 的功能。

不过需要注意数量，和节点数量保持一致；如果节点减少，需要手动调整 replicas；

尽管使用 Deployment 能近似实现 DaemonSet 的要求，但在功能完整性和自动化程度上，还是比不上原生的 DaemonSet。

4.4.1.18 怎么理解 Docker、K8s、容器

- ✧ 容器，是镜像的运行实例，隔离进程，共享主机内核，秒级启动。
- ✧ Docker，是容器化工具，将应用及其依赖打包成轻量级、可移植的镜像。
- ✧ Kubernetes（K8s），是容器编排系统，自动化部署、扩展和管理多容器应用，提供负载均衡、自愈等功能。

关系：Docker 创建容器 →K8s 管理集群中的容器。

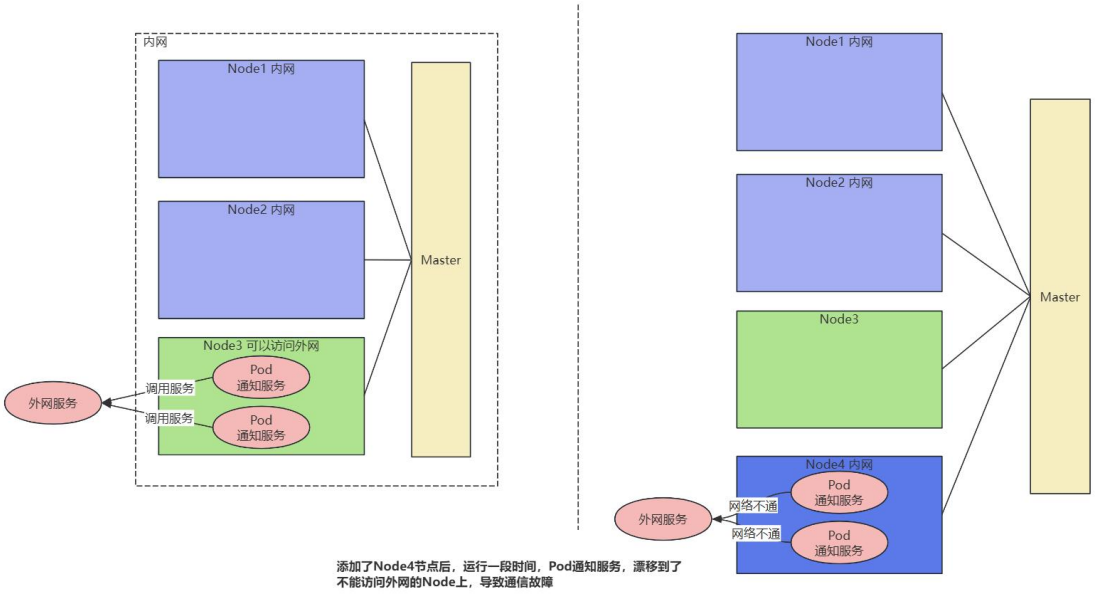
4.4.2 项目常问

4.4.2.1 K8s 在生产（现网）所遇到的印象深刻的故障

案例一 网络问题

迁移到 K8S 集群遇到的最多的就是网络问题，当时系统需要调用几个第三方的地址(百度 OCR、电子签等等)，需要开通 K8S 集群 node 节点到第三方地址的网络权

限，后来发现服务调不通了，排查发现是 K8S 集群加了 node 节点，新加的 node 节点没有开通到第三方的网络权限，**应用是调度到了新加的 node 节点上所以就调不通了。**



类比一个例子，一个教室里，只有老师能够访问外网权限。这个时候，来了一个任务，要求调一下外部的接口。但是呢，无意之间把这个任务派给学生了。最后，这个学生的上面始终访问不了。所以这个**跟外网交互的任务必须给老师去做。**

案例二 Secret 卷内容为空导致服务通信失败

有一次，我们安全上要做一个整改，引入了一个 secret 卷，用来保存初始化密码。密码的话，要求是至少 8 位，至少 4 种字符。

以下是关键部分的 Python 代码

```
import random
import string

def generate_password():
    # 定义字符集
    digits = string.digits                  # 数字 0-9
    lower = string.ascii_lowercase    # 小写字母
    upper = string.ascii_uppercase    # 大写字母
    special = '!@#$%^&*()_+-=[]{}|;:,.<>?'    # 特殊字符

    # 确保每种类型至少有一个字符
    password = [
        random.choice(digits),
```

```
            random.choice(lower),
            random.choice(upper),
            random.choice(special)
        ]

        # 填充剩余 4 个字符（从所有字符集中随机选择）
        all_chars = digits + lower + upper + special
        password += random.choices(all_chars, k=4)

        # 打乱顺序
        random.shuffle(password)

        # 组合成字符串
        return ''.join(password)

# 示例使用
print(generate_password())  # 输出类似：aB3!xY7@
```

这个改造完成之后，我们在一个环境上部署成功了。后来又布了一个环境。在这个环境当中，我们发现有一个页面前端能够加载出来，但是后端的接口每次调用隔壁部门的鉴权模块时候就报错。这个时候，测试找到了我们的开发同学，开发同学找到了我们的运维，一起去定位。

我们运维定位，发现每个 Pod 都是 running，没啥问题。让开发同学一起排查。

这时候开发同学发现，每次调接口后台日志服务就会报错，说可能是鉴权模块儿那边的问题，于是让我们去找鉴权模块。鉴权模块的同学说一起来看一下。他们上传了一个 Java 内存查看工具，发现我们每次调用的时候，**接口有一个入参为空。**

于是，鉴权同学退下了，开发同学继续在 pod 里面查看，发现挂载的 secret 的下面有文件，但是文件里面内容**真的是为空**。这个时候，我们又找到了公共组的安全开发，问他们这个卷怎么是空的？**他们说这不可能！因为另外一个环境刚布好，都没啥问题，你们这个环境怎么会出问题呢？**

于是又拉上了我们运维的基座同学一起看了初始化脚本，最后发现在初始化创建的时候，日志里打印了一行信息，**提示这个卷创建出来，但是里面是空**。这个时候我们发现。原来是 Python 代码里异常处理封装的特别好，以至于创建出来失败也没有报错。后来我们就去检查，发现它创建出来的密码包含有特殊字符反斜杠，反斜杠在开发当中是转义的意思，所以导致，8 位字符少了 1 位密码，最后对不上。

我们才发现，是因为里面包含的有特殊字符，而另外一个环境没有问题，是因

为那个环境当中的密码随机生成的时候，不含有反斜杠特殊字符。

最后拉到了我们的安全工程师，拉到了我们架构师。认为是测试的时候测试用例不完备，导致**随机取值的时候没有测出来**。于是，我们要求必须要做全量测试，**把所有的特殊字符加上大小写，全部放在了密码卷里做测试**，最后发现有些字符干脆就可以把它给踢掉，简单搞让他百分之百不会报错，最后我们这个 Bug 才算修复掉。

案例三 Pod 全量重启，导致服务拉起失败

在老版本当中，有一次，我们有个同事，修改了一个证书 Secret，触发了所有 Pod 的重启，但是当时环境部署的是标准版私有云，资源规格有限。这个时候，200 多个 Pod 同时重启，**全部是 pending**，起不来了。

拉上了我们的运维同事去排查，发现 Node 节点提示资源不足，无法初始化 Pod。后来我们把环境上所有的 Pod 全部停止了。然后每次只拉取一个模块，大概 20 个 Pod，每次等待 3 分钟，就这样，拉取了 10 次左右，把全部的 Pod 起来了。

最后，这个问题一直持续了半个小时，才解决完，环境才恢复。好在当时就是做 Secret 卷更新，所以没有造成太大事故，但是影响时间太久，后来让我们的开发去优化方案了，对于公共挂卷，不能全部秒级重启，运维组长和架构师沟通，后面版本改成滚动升级，来解决。每次只更新 10 个 Pod 左右。

4.4.2.2 都用到了哪些探针，是哪个服务发起的探针

存活探针、就绪探针、启动探针。

在 K8s 中，探针是由 kubelet 服务发起的。kubelet 是 K8s 集群中每个节点上运行的主要代理，它负责管理和监控节点上的容器和 Pod。当定义了 Pod 的探针（包括存活探针 livenessProbe、就绪探针 readinessProbe 和启动探针 startupProbe）时，kubelet 会按照配置的参数定期执行这些探针，以检查容器的状态。

4.4.2.3 假设集群中一个 Node 负载已经非常高了，这个时候要 kill 掉一个 pod，是怎么实现的？

第一种，是基于资源请求和限制的驱逐机制

Kubernetes 中的 kubelet 会定期检查，会根据 Node 的资源使用情况和 Pod 的资源请求与限制，按照优先级和 QoS（服务质量）等级来选择要驱逐的 Pod；

第二种，基于节点压力的 Pod 驱逐

Kubernetes 根据 Node 的压力指标（如磁盘压力、PID 压力等）来驱逐 Pod，当 Node 出现磁盘空间不足、PID 数量达到上限等情况时，kubelet 会自动驱逐一些 Pod 以缓解压力。

4.4.2.4 有没有遇到过调度不均匀的情况？

有三个思路。

第一个思路就是对于资源使用太多的 Node 节点，打上污点，阻止 Pod 调度上来；

第二种是结合节点标签和 Pod 标签选择器，将 Pod 调度到性能充足的那些 Node 上面；

第三种就是既然容易出现调度不均匀，在后期资源规划的时候优化调整，统一 Node 节点的资源规格；

4.4.3 个人职责

模板 1：集群部署与维护

- 参与 Kubernetes 生产集群的部署和配置工作
- 维护集群核心组件（etcd、kube-apiserver 等）的运行状态
- 执行集群版本升级和补丁更新
- 监控集群健康状态，处理节点异常问题
- 配置和维护集群网络插件（Calico/Flannel 等）

模板 2：应用部署与管理

- 使用 Helm Chart 管理应用部署和配置
- 编写和维护 Kubernetes 资源清单（Deployment/Service 等）
- 配置 HPA 实现应用自动扩缩容
- 管理应用命名空间和资源配额
- 处理应用部署过程中的各种问题

模板 3：存储与网络配置

- 配置和管理 PersistentVolume/PersistentVolumeClaim
- 维护 StorageClass 和 CSI 驱动
- 配置 Ingress 控制器和服务暴露

- 优化服务网络性能和可靠性
- 处理存储相关的故障和性能问题

模板 4：监控与日志

- 部署和维护集群监控系统（Prometheus 等）
- 配置告警规则和通知渠道
- 收集和分析容器日志
- 监控集群资源使用情况
- 参与性能调优和容量规划

模板 5：安全与合规

- 实施 RBAC 权限控制策略
- 配置网络策略（NetworkPolicy）
- 处理安全漏洞和合规要求
- 维护集群证书和密钥
- 参与安全审计和加固工作

4.5 Prometheus 监控

4.5.1 面试必问

4.5.1.1 Prometheus 监测 K8s Pod 的时候怎么做服务发现的

Prometheus 利用 Kubernetes 的 API Server 来获取集群中的资源信息。

1、节点（Node）发现。在 Prometheus 配置文件中添加相关配置，指定 role 为 node，这样 Prometheus 会定期从 API Server 拉取节点信息，并为每个节点生成相应的监控目标。

2、Pod 发现。以 Pod 为目标进行服务发现时，Prometheus 会查找所有符合条件的 Pod，并根据配置采集其暴露的指标。可以通过标签选择器筛选出需要监控的 Pod。

换句话说，你配置了 role，它就可以定期去调接口，发现节点和 pod 信息；

4.5.1.2 Pormetheus 的架构，有哪些组件，怎么监控的？可以监控哪些指标

Prometheus 是一款开源的系统监控和时间序列数据库，其架构主要由以下组件构

成：

1. Prometheus Server

负责收集和存储时间序列数据。它通过 HTTP 协议从配置好的目标端点拉取数据，并按照一定的规则进行存储和处理。可以配置多个 Prometheus Server 以实现高可用和分布式部署。

2. Exporter

数据采集组件，用于将各种系统、应用程序或服务的指标数据转换为 Prometheus 能够识别的格式，并提供给 Prometheus Server 进行采集。

3. Pushgateway

数据中转组件：主要用于接收短时间运行的任务或批处理作业的指标数据。由于这些任务可能在 Prometheus Server 拉取数据之前就已经结束，所以需要将指标数据主动推送到 Pushgateway，然后 Prometheus Server 再从 Pushgateway 拉取数据。

4. Alertmanager

告警管理组件：负责接收 Prometheus Server 发送的告警信息，并根据配置的告警规则进行处理，如分组、抑制、发送通知等。

5. Web UI

可视化组件：Prometheus 自带了一个简单的 Web UI，用于查看和查询指标数据。用户可以在 Web UI 中输入查询语句，以图形化或表格的形式展示指标的变化趋势。

6. Rules

规则组件：包括记录规则和告警规则。记录规则用于预先计算和存储一些常用的复杂查询结果，以提高查询性能；告警规则用于定义触发告警的条件和阈值，当指标数据满足告警规则时，Prometheus Server 会将告警信息发送给 Alertmanager。

4.5.1.3 Prome 怎么监控 K8s 系统？Prome 怎么拿到 Pod 的指标

1、Prometheus 监控 K8s 系统并获取 Pod 指标，通常需要以下几个步骤：

a.部署 Kube-State-Metrics，通过 Helm Chart 或直接使用 YAML 文件进行部署；

b.部署 Node Exporter，一般在每个节点上以 DaemonSet 的方式运行，以便监控所有节点；

2、配置 Prometheus Server 的服务发现，使得 Prometheus 自动采集 Metrics 暴露的指标；

3、Prometheus 通过 Kube-State-Metrics 获取 Pod 的详细指标。

4.5.2 项目常问

无。

4.5.3 个人职责

模板 1：监控系统部署与维护

- 负责 Prometheus 在 Kubernetes 集群中的部署和配置
- 维护 Prometheus Operator 及相关 CRD 资源
- 管理 Alertmanager 配置，设置告警路由规则
- 监控 Prometheus 自身健康状况和资源使用
- 处理 Prometheus 存储和抓取性能问题

模板 2：指标采集与配置

- 配置 ServiceMonitor/PodMonitor 采集应用指标
- 编写和维护 PromQL 查询语句
- 为应用添加合适的监控指标暴露端点
- 优化指标采集频率和保留策略
- 处理指标丢失或异常问题

模板 3：告警管理

- 设计和配置合理的告警规则
- 维护告警分级和静默策略
- 设置告警通知渠道（邮件/钉钉/Webhook 等）
- 分析告警误报和漏报原因
- 参与告警值班和响应流程

模板 4：可视化与报表

- 配置和维护 Grafana 仪表盘
- 设计业务和技术指标可视化方案
- 定期生成监控分析报告
- 优化仪表盘加载性能
- 维护监控数据权限控制

模板 5：监控系统优化

- 优化 Prometheus 存储和查询性能
- 配置远程写入和联邦集群

- 实现监控系统高可用部署
- 参与监控系统容量规划
- 探索新的监控方案和最佳实践

4.6 CI/CD 项目

4.6.1 面试必问

4.6.1.1 讲一讲你们公司从开发给代码，到打包镜像，到环境部署，业务上线的全部过程吧。

4.6.2 项目常问

无。

4.6.3 个人职责

模板 1：流水线设计与实现

- 参与 CI/CD 流水线的设计与搭建工作
- 编写和维护 Jenkinsfile/GitLab CI 配置文件
- 配置多环境部署策略（开发/测试/生产）
- 实现自动化构建、测试和部署流程
- 处理流水线执行过程中的各种问题

模板 2：代码质量管理

- 集成代码静态分析工具（SonarQube 等）
- 配置单元测试和集成测试自动化执行
- 维护代码覆盖率统计和报告
- 实施代码审查自动化流程
- 处理代码质量相关的告警和阻断

模板 3：制品管理与发布

- 维护制品仓库（Nexus/Artifactory 等）

- 管理构建产物的版本和依赖关系
- 实现自动化版本发布流程
- 维护发布回滚机制
- 处理制品安全扫描相关问题

模板 4：环境配置管理

- 使用 Ansible/Terraform 管理部署环境
- 维护多环境配置和差异化部署
- 实现环境自动准备和清理
- 处理环境配置冲突问题
- 参与环境治理和标准化工作

模板 5：流程优化与创新

- 探索和引入新的 CI/CD 工具和技术
- 优化现有流水线执行效率
- 参与 DevOps 实践和流程改进
- 协助团队适应 CI/CD 工作方式
- 维护 CI/CD 相关文档和知识库

4.7 ERP 项目

4.7.1 面试必问

无。

4.7.2 项目常问

4.7.2.1 介绍一下你们的 ERP 是干什么的

(蓝云) 商贸 ERP 系统针对中小型商贸企业开发设计的企业资源计划管理系统。通过 ERP 系统的管理，企业可以有效管理经营各个环节中的业务数据。企业数据经过收集汇总后，为各级终端用户提供日常业务信息管理、业务流程执行等日常办公服务辅助平台，辅助本职能部门出具预案决策，提高整体企业内部业务工作效率。

4.7.2.2 能不能简单聊一聊这个项目的背景

这个项目是为××（名称自己构思，为防止同学们面试期间撞车，此处不提供公司名称，但是此名称需要同学们调查一下这个公司是否真实存在，在百度上搜索一下，例如武汉商贸公司，列出来一些名字，不要找第一页的名字，最好向后翻几页）商贸公司做的，公司原有的项目是一个覆盖地区为省的平台，现在公司规模扩张，全国各地建分公司，省内管理相对来说比较有局限性，加上全国全联网，采购、销售、货运、仓储、配货、财务核算各方面发生了很大的变化，所以重新起了一套新的平台。

这个项目可以说算是二次开发，但是业务里程变化蛮大的。尤其是加上了一些数据分析，数据挖掘的东西，而且报表的统计数据量也比之前的大很多，复杂度高很多。这个项目开发主要是从 CRM、进销存、仓储、财务四个方面做了大的改动，基础平台方面多添加了一些功能，比如像合同啊，报销啊这些东西不再以网点为单位了，改成全国统一编排。

4.7.2.3 ERP 系统所涉及的业务模块和流程

本系统包括了：基本信息管理、人事管理、采购管理、销售管理、库存管理、统计分析、权限管理等模块。

各个模块说明：

- **基本信息管理：**管理商品类型、商品信息、供应商信息、客户信息以及仓库信息等功能；
- **人事管理：**管理员可以对部门和员工进行管理；
- **采购管理：**主要是采购员对缺货商品进行采购操作包括退货操作；
- **销售管理：**销售员对商品进行销售订单处理以及销售订单退货管理；
- **库存管理：**用户可以操库存管理页面进行库存查询以及库存变动记录查询；
- **统计分析：**主要分几块：销售统计报表分析、库存预警报表实现、盘盈盘亏实现等；
- **权限管理：**根据不同的角色，分配不同的权限，可以查看相对应可以查看得内容；

4.7.2.4 如果用户反馈 ERP 系统无法登录，你会如何排查

我会按照以下步骤排查：

检查网络：确认用户网络是否正常，能否 Ping 通服务器。

验证账号：检查账号是否被锁定或密码过期（如 AD 域控或数据库用户表）。

查看日志：检查应用日志（如 Tomcat 的 catalina.out）是否有登录相关的错误（如 Connection refused 或 Invalid credentials）。

服务状态：确认 ERP 服务是否正常运行（如 systemctl status tomcat）。

数据库连接：检查数据库连接是否正常（如 Oracle 的 lsnrctl status）。

4.7.2.5 你如何监控 ERP 系统的运行状态

我主要使用以下工具和方法：

基础监控：Zabbix 或 Prometheus 监控服务器 CPU、内存、磁盘、网络指标。

服务监控：通过脚本或工具（如 ps-ef|grepjava）检查关键进程（如 Tomcat、Nginx）是否存活。

日志监控：使用 ELK（Elasticsearch+Logstash+Kibana）或 grep 命令筛选错误日志。

数据库监控：通过 OracleEnterpriseManager 或 MySQLWorkbench 查看数据库连接数和慢查询。

4.7.2.6 你处理过哪些 ERP 系统的常见故障

我处理过的常见故障包括：

服务崩溃：Tomcat 或 Nginx 进程异常退出，通过重启服务解决。

数据库连接满：Oracle 或 MySQL 连接数达到上限，通过增加连接池或优化 SQL 解决。

文件上传失败：检查磁盘空间是否不足，或 Nginx 上传文件大小限制。

报表加载慢：优化 SQL 查询或增加数据库索引。

用户权限问题：调整 AD 域控或数据库账号权限。

4.7.2.7 你如何备份 ERP 系统的数据

我参与的备份策略包括：

数据库备份： MySQL：通过 mysqldump 导出数据，并配置定时任务（Crontab）。

文件备份：使用 rsync 或 tar 命令备份应用日志和配置文件。重要文件上传至云存储（如阿里云 OSS）。

验证备份：定期恢复测试，确保备份文件可用。

4.7.3 个人职责

模板 1：系统运维方向

- 负责 ERP 系统的日常监控、维护和故障排查，确保系统稳定运行。
- 处理用户提交的运维工单，解决账号权限、数据查询等问题。

- 协助数据库管理员进行 Oracle 数据库的备份、恢复和性能监控。
- 参与服务器资源管理，包括 CPU、内存和磁盘使用情况的定期检查。
- 编写自动化脚本简化日常运维任务，如日志清理、定时备份。

模板 2：系统运维方向|系统升级

- 协助搭建测试环境，部署 Java 后端服务（Tomcat）和前端静态资源（Nginx）。
- 监控系统升级过程中的服务状态，记录异常日志并反馈给开发团队。
- 参与数据库迁移工作，验证数据完整性和一致性。
- 配合测试团队完成功能测试和性能测试，确保升级后系统稳定。
- 整理运维文档，包括部署手册和常见问题解决方案。

模板 3：云服务与监控方向

- 负责 ERP 系统在阿里云上的 ECS 实例管理，包括安全组配置和实例监控。
- 使用 Prometheus 和 Grafana 搭建系统监控平台，跟踪关键指标。
- 处理云服务器资源告警，分析 CPU、内存和磁盘使用情况。
- 协助完成数据库（RDS）的备份策略制定和灾备演练。
- 参与运维自动化工具的研究和测试，如 Ansible 和 Terraform。

模板 4：故障排查与用户支持方向

- 负责 ERP 系统的用户账号管理（AD 域控）和权限配置。
- 处理用户反馈的系统问题，如登录失败、单据提交异常等。
- 分析系统日志，定位故障原因并协调开发团队解决。
- 参与服务器和网络设备的日常巡检，记录运行状态。
- 协助编写运维手册和常见问题解决方案，供团队参考。

模板 5：故障排查与用户支持方向

- 参与系统压测，记录 TPS 下降时的资源占用情况。
- 配合数据库管理员优化 SQL 查询，减少慢查询对系统的影响。
- 整理性能优化过程中的技术文档和测试报告。
- 协助测试团队验证优化后的系统稳定性。

4.8 信创项目

4.8.1 面试必问

4.8.1.1 国产化数据库了解哪些

达梦数据库（DM）：由达梦数据库有限公司开发，其前身是华中科技大学数据库与多媒体研究所。是国内较早的数据库产品之一，支持多种操作系统和硬件平台，数据类型丰富，SQL 语言功能强大，具有高可靠性、高并发性和高安全性等特点。适用于金融、电信、政府等对数据安全性和稳定性要求较高的领域。

金仓数据库（KingbaseES）：北京人大金仓信息技术股份有限公司研发，由中国人民大学及一批专家于 1999 年发起创立。产品稳定运行于多种主流操作系统平台，与国内主要操作系统和中间件厂商有产品兼容性认证，安全级别达到《GB/T 20273 - 2006 信息安全技术 数据库管理系统安全技术要求》第三级。广泛应用于政府、军队、电力、金融等领域。

南大通用 GBase：南大通用数据技术股份有限公司是国内领先的数据库产品和解决方案供应商。GBase 8a 是支撑大数据快速分析的新型数据库，可实现全数据存储管理和高效分析；GBase 8t 是与世界技术同级的国产事务型通用数据库系统，能在多数场景中替代 Oracle；GBase 8m 是处理速度快的内存数据库；GBase 8s 是具有安全防护性能的关系型数据库。产品应用于政府、金融、电信等多个行业。

OceanBase：由蚂蚁集团完全自主研发，始创于 2010 年。采用自研的一体化架构，兼顾分布式架构的扩展性与集中式架构的性能优势，用一套引擎同时支持 TP 和 AP 的混合负载，具有数据强一致、高可用、高性能、在线扩展、高度兼容 SQL 标准和主流关系数据库、低成本等特点。主要应用于金融、政府、运营商等行业，能支撑大规模电子商务、金融等场景下的高并发、分布式事务等复杂需求。

GaussDB：华为推出的关系型数据库，支持 PB 级别的大规模数据存储和处理。采用分布式架构和云原生技术，具备企业级复杂事务混合负载能力，同时支持分布式事务，同城跨 AZ 部署，数据 0 丢失，支持 1000 + 的扩展能力。具有高性能、高可用、高安全性等优点，适用于云计算、大数据、人工智能等领域，在华为云生态中占据重要地位，广泛应用于政府、金融等行业。

4.8.1.2 国产操作系统用过哪些

银河麒麟：由麒麟软件有限公司主导研发（其前身是国防科技大学）。面向国家安全战略需求，拥有完全自主知识产权。基于 Linux 内核，采用微内核、模块化设计，支持多种 CPU 架构，具备良好的兼容性与稳定性，通过多重安全机制构建安全防护体系，广泛应用于政务、军事、金融、能源、交通等关键领域。

统信 UOS（统一操作系统）：由统信软件技术有限公司打造，整合了多家国产厂商的技术资源。支持多种国产 CPU 平台，提供专业版、家庭版、社区版和服务器操作系统等多种版本。系统设计符合国人审美和习惯，安全可靠，全面兼容国内外主流软硬件，确保用户无缝迁移，广泛应用于政府、企业、教育、个人等多个领域。

华为欧拉（openEuler）：由华为技术有限公司开发，是面向服务器的 Linux 发行版，由华为创建并贡献给 openEuler 开源社区。具有高稳定性，经过严格测试和优化，确保系统稳定运行，广泛兼容多种处理器架构和信创芯片，通过开放的社区形式与全球开发者共同构建软件生态体系，广泛应用于服务器、云计算、边缘计算等场景。

4.8.2 项目常问

无。

4.8.3 个人职责

无。