

NEW PLAN

软件高级架构师

• 一段新征程 •





01

论点寻找

论点的提取不是拍脑袋想三个点，而是要根据论文的题目以及你项目的特点来选择提取维度，并且切记在论述的时候要贴合论文题目的第三问，常见的论点提取方式有以下几种，你必须根据题目要求灵活选择：

方式一：按技术架构分层提取

微服务架构：

- 接入层：API网关的统一认证与流量控制设计
- 服务层：基于领域驱动设计的服务拆分与治理
- 数据层：分布式事务与最终一致性保障机制。

最匹配的第三问场景：侧重"架构设计"或"技术实现"，如"如何基于微服务架构进行软件设计实现"

方式二：按项目实施阶段/生命周期提取

Serverless：

- 需求分析阶段：基于事件驱动的用例建模与函数边界划分
- 架构设计阶段：FaaS与BaaS的协同设计（函数编排与状态管理）
- 开发实施阶段：冷启动优化与可观测性埋点实践

最匹配的第三问场景：明确提到了阶段要求，如"Serverless架构在需求分析、设计、开发中的应用"

方式三：按技术特性维度提取

云原生数据库：

- 弹性维度：存储计算分离架构下的在线扩缩容方案
- 可用性维度：多可用区部署与故障自动切换机制
- 可观测性维度：全链路SQL追踪与智能诊断平台构建

最匹配的第三问场景：侧重某项技术的优势体现或特性落地，如"云原生数据库的技术优势及架构设计"

方式四：按业务流程阶段提取

秒杀场景：

- 前置风控阶段：实时反欺诈与身份核验机制
- 核心交易阶段：多方资金划转的分布式事务协调
- 后置对账阶段：异步清算与差错冲正机制

最匹配的第三问场景：针对特定场景的技术方案，如秒杀、支付、物流、订单等特定业务场景

核心原则：论点必须与第三问严丝合缝地呼应

第三个子问题和论点的关系：既可以直接拿来做论点，也可以自己找论点但是必须要包含子问题

例如：第三个子问题问的是：**Serverless**在需求分析、设计、开发中的应用

很明显这个问题就直接包含了几个点，你就按这几个点来作为论点来叙述即可，如果你对这几个点不熟悉，觉得按照它们来写你写不了，那也可以自己找论点，但是你在论述的过程中，必须要包含分析、设计和应用，只不过在篇幅中，可以根据你自己的情况进行弱化一些不擅长的地方。

示例1：详细论述云原生数据库的核心技术优势，以及架构设计中如何体现云原生数据库的技术特性

- 提取逻辑：既然问的是技术优势，那就按云原生数据库的核心特性（弹性、高可用、容灾备份）来切分
- 弹性优势体现：存储与计算分离架构下的在线扩缩容设计
- 高可用优势体现：基于云原生数据库的同城双活架构设计
- 容灾备份优势体现：基于分布式快照的秒级备份与分钟级恢复机制

示例2：说明基于云原生数据库的架构选型依据、落地过程中的关键难点及应对措施

- 提取维度就要改为：选型对比→迁移难点→优化措施
- 选型：从**Oracle**到云原生数据库的选型博弈——对比**PolarDB**、**TiDB**在兼容性、成本、生态上的权衡，说明选择依据
- 难点：存储过程改造与**SQL**方言适配
- 措施：双轨并行运行与数据一致性校验

02

论文预测

论题一：云原生微服务架构与服务治理

预测理由：云原生是企业转型的绝对核心，但考题已从宽泛概念转向具体技术点。2025年5月考了事件驱动（微服务通信模式），2025年11月考了Serverless（微服务演进形态），那么微服务本身及其治理技术就是自然延伸。**服务网格、可观测性**等深化技术尚未独立命题，完全有可能考到。

涉及考察点：

- 微服务拆分原则（DDD领域驱动）
- 服务注册与发现（Eureka/Nacos）、配置中心（Apollo/Nacos）、API网关（Spring Cloud Gateway/Kong）
- 服务治理核心：熔断、降级、限流、负载均衡
- **可观测性体系：链路追踪（SkyWalking/Jaeger）、指标监控（Prometheus）、日志聚合（ELK/Loki）**
- **服务网格进阶：Sidecar模式（Envoy）、Istio流量管理、零信任安全（mTLS）**
- 容器化（Docker）与Kubernetes编排（Deployment/Service/Ingress/HPA自动扩缩容）

论题二：数据库架构与多模数据架构

预测理由：数据架构每年必考，2023-2025年已覆盖数据集成、Lambda架构、多模型数据库、云原生数据库。未来聚焦数据库技术深化与多模统一，向量数据库作为AI时代的新型数据存储，与传统关系/文档/图数据库共同构成多模架构。

可能考察点：

- 云原生关系数据库(TiDB/PolarDB/OceanBase)，一般应用在一些传统的领域，如交易、ERP、核心账务
- 图数据库(Neo4j/TuGraph)，一般应用在风控、社交关系、知识图谱等领域
- 向量数据库(Milvus/Weaviate/PGVector)，一般应用在RAG、推荐、相似度检索
- 实时采集：Kafka/Pulsar消息队列，CDC技术
- 流处理引擎：Flink窗口计算、状态管理 (Checkpoint)
- 实时OLAP：ClickHouse/Doris列式存储，预聚合优化

论题三：高并发系统性能优化场景

预测理由：考题明显转向场景解决方案。2024-2025年已考分布式事务、性能测试、秒杀场景，性能优化作为通用工程能力是自然延伸。

涉及考察点：

- 瓶颈分析：CPU/内存/IO/网络定位，火焰图分析，慢查询优化
- 多级缓存：本地缓存→分布式缓存（Redis）→CDN，缓存一致性
- 异步化：RocketMQ/Kafka削峰填谷，最终一致性，死信队列
- 系统韧性：熔断降级（Sentinel）、限流算法（令牌桶/漏桶）、舱壁隔离
- 容量保障：全链路压测、弹性伸缩、混沌工程

论题四：论AIGC技术对软件架构设计的影响与应用

预测理由：论文以后肯定会拿出一道题目来考未来技术，比如25年5月考了AI的其中一个应用方向（测试），那么下半年是否会考现在很火的大模型或者是某个具体的应用技术。

可能考察点：

- 如何利用AIGC进行设计文档辅助生成、技术方案辅助评审。
- 探讨AI编程助手对开发流程和架构模式的影响。
- 设计基于大模型的智能编程或系统运维助手。



2026年11月论文初步预估



知乎知学堂

论题一（架构题可能性1）：云原生微服务架构与服务治理

预测理由：微服务、云原生已经连续多年出现，但之前更多考的是架构风格、通信模式、Serverless。真正把“可观测性”作为独立论文题，尚未大规模出现。

云原生架构复杂度上升后，可观测性已经成为架构师必须掌握的核心能力，而且它和微服务治理、高并发、分布式系统都强相关，非常适合作为论文题。

涉及考察点：

- 可观测性三大支柱：Metrics、Logging、Tracing
- 链路追踪：SkyWalking、Jaeger、OpenTelemetry
- 指标监控：Prometheus、Grafana、告警体系
- 日志聚合：ELK、Loki、日志标准化
- 可观测性数据采集、传输、存储、分析、展示
- 与微服务治理结合：熔断、限流、降级的观测
- 可观测性平台的高可用、海量数据处理

切入点建议：

你可以写一个微服务或分布式系统项目，因服务数量增加导致排障困难，设计统一可观测性平台，进行采集数据、监控指标、链路追踪、日志聚合，最终实现故障定位时间大幅下降。



2026年11月论文初步预估



论题一（架构题可能性2）：架构评估

预测理由：ATAM/SAAM间隔7年严重超期 + 2026上全是实践题需要方法论平衡 + 教材核心知识点。

涉及考察点：

- **SAAM**评估方法：场景生成、架构描述、场景分类与优先级排序
- **ATAM**核心流程：效用树构建（质量属性→属性分类→具体场景）、架构权衡、风险点/敏感点/权衡点识别
- **CBAM**成本效益分析：基于ROI的架构决策、成本与质量属性的量化权衡
- 质量属性场景六要素：刺激源、刺激、环境、制品、响应、响应度量
- 实际评估流程：评估准备→场景分析→架构分析→评审报告→架构改进
- 与具体项目结合：如何识别性能瓶颈、可扩展性风险、安全漏洞

切入点建议：

可以写一个某某交易系统的架构评估项目，采用 **ATAM**对系统进行质量属性评估。重点写如何构建质量属性效用树、定义具体质量属性场景（如高并发下交易延迟、故障恢复时间、可修改性等），识别架构中的风险点、敏感点与权衡点，并基于评估结果提出架构优化方案（如引入分布式缓存、消息队列削峰、服务降级等），体现评估方法对架构决策和改进的指导作用。



2026年11月论文初步预估



知乎学堂

论题二：论实时数据处理与湖仓一体设计

预测理由：数据题铁律 + Lambda已考 + Kappa是大数据架构的自然演进。Flink、Kafka、实时数仓是架构师核心技能栈，命题价值极高。

可能考察点：

- 图数据库(Neo4j/TuGraph)，一般应用在风控、社交关系、知识图谱等领域
- 实时采集：Kafka/Pulsar消息队列，CDC技术
- 流处理引擎：Flink窗口计算、状态管理 (Checkpoint)
- 实时OLAP：ClickHouse/Doris列式存储，预聚合优化
- 湖仓一体：Iceberg、Hudi、Delta Lake
- 实时数据与离线数据融合

切入点建议：

可以写一个实时风控、实时报表或实时推荐系统，设计 CDC+Kafka+Flink+ClickHouse 的实时数仓架构，解决传统 T+1 数据延迟问题。重点写分层设计、状态管理、Exactly-Once 语义、数据质量保障。



2026年11月论文初步预估



知乎学堂

论题三：论高可用与业务连续性场景

预测理由：考题明显转向场景解决方案。2024-2026年已考分布式事务、性能测试、秒杀场景、高并发性能优化，高并发之后，架构师最经典的下半篇就是高可用、容灾多活，因为它直接承接 26 上的高并发论题。

涉及考察点：

- 可用性指标：SLA / SLO / RTO / RPO；
- 架构原则：无状态化、冗余、故障转移、心跳、选主；
- 部署架构：同城双活、异地多活、单元化；
- 流量调度：DNS、GSLB、网关流量切换；
- 数据同步：主从、分库分表、分布式数据库、数据一致性；
- 稳定性保障：熔断、降级、限流

论题四：论AIGC技术对软件架构设计的影响与应用

预测理由：论文以后肯定会拿出一道题目来考未来技术，比如25年5月考了AI的其中一个应用方向（测试），26年5月也考了大模型+移动端测试，所以后面老师觉得很有可能出AI+软件工程某个生命周期的命题。

可能考察点：

- AIGC对开发流程的重构：AI辅助需求分析、AI生成设计文档、AI辅助架构决策
- 安全与合规：模型幻觉防护、数据隐私保护、提示词注入防御、AIGC内容审核
- 与现有架构融合：如何在微服务/事件驱动架构中嵌入AI能力、AI服务的弹性伸缩设计

03

论文鉴赏



论文鉴赏评分标准



1.摘要： 时间+项目简介+个人岗位+工作职责+【投入+历时+团队介绍】+成功交付客户好评+结合具体题目说明本文结构

2.背景： 项目背景及过渡部分建议350-450字， 只跟项目有关， 可以适配任何论文题目。

3.第一个子问题： 时间+项目简介+个人岗位+工作职责+【投入+历时+团队介绍】+成功交付客户好评、简要介绍几个项目的功能模块、项目所用技术（不要对着摘要全部抄袭， 注意看看第一个子问题有没有其他额外的问题需要你回答）。

-----过渡段/句-----

4.第二个子问题： 结合上下文综合你的项目一起来回答这个简单题， 不要单独把问题列出来回答， 注意看题目有没有在第三问给定论点， 如果没有论点， 那么在第二个子问题解答的时候， 尽可能把你解答的方向作为你第三个子问题的论点， 这样子文章符合总分的原则。

-----过渡段/句-----

5.第三个子问题： 一般来说， 第三个子问题我们都会划分为3个点并且分段进行回答， 回答的时候， 先针对论点进行解释， 然后再记叙你项目和该论点有关联的地方， 再说明你项目之前存在的问题， 你做了哪些方案， 这些方案你如何做取舍的， 最后选择了哪一个方案， 涉及到了哪些技术， 最后结果如何， 注意： 在这里不要说你项目采用了这些方案之后存在的不足， 放在结尾写。

6.结尾： 项目上线及运行效果（用可以量化的数据来说明）、客户评价、项目收获、项目不足和解决思路（不足和解决思路怎么写， 需要看字数的， 如果前面的字数已经有2000字了， 这里不写不足和解决思路都可以， 如果字数不够， 那么你必须写不足和解决思路， 水个2,300字出来）



范文3：论面向对象建模



论面向对象建模及其在系统架构设计中的应用（AI生成的题目）

面向对象建模（**Object-Oriented Modeling, OOM**）是一种通过抽象、封装、继承和多态等原则，将现实世界问题转化为软件系统模型的系统化方法。它通过类图、用例图、序列图等 **UML** 工具描述系统静态结构与动态行为，支持从需求分析到代码实现的全生命周期设计。面向对象建模在复杂业务系统、分布式架构及领域驱动设计（**DDD**）中具有重要价值，但需平衡灵活性、可维护性与性能优化等挑战。请围绕“面向对象建模及其应用”论题，依次从以下三个方面进行论述：

- 1.概要叙述你参与管理和开发的软件项目，并说明项目中采用面向对象建模的核心目标与业务需求。
- 2.简要分析面向对象建模的关键技术，并阐述其在系统架构设计中的核心价值与局限性。
- 3.具体阐述你在项目中如何应用面向对象建模方法，包括需求分析、模型设计、技术实现及效果验证，并说明遇到的典型问题与优化方向。

2018 年 3 月 1 日至 12 月 20 日，我参加了“数据安全访问平台”项目的开发，担任系统分析员的工作。该项目是某行业用户“数据中心二期”建设的主要内容，目标是：建立数据统一访问接口及其使用标准，规范、约束和审计数据应用访问数据库的行为，对数据应用提供强制审计的技术手段。由于该系统是所有应用的基础平台，对系统的可靠性与性能有较高要求，同时由于没有成熟的现有系统作为参照，该项目存在较高的风险。

本文结合作者实践，讨论了在项目中基于 UML 的需求分析。我们使用用例图描述用户与系统的交互；使用类图描述系统的核心概念；使用部署图描述系统的网络部署；使用活动图描述系统的应用流程。由于采用了 UML 中的多种技术，使得我们能从多个方面完整的把握需求，有效的保证到了需求工作的质量。最后，分析了需求工作中存在的问题和改进的方法。

2018 年 3 月 1 日至 2018 年 12 月 20 日，我参加了“数据安全访问平台”项目的开发，担任系统分析员的工作。“数据安全访问平台”是某行业用户“数据中心二期”建设的主要内容。在一期建设中已建成数据的统一存储和统一分发框架。但主要存在以下问题：无法获得应用用户对数据库的操作日志；开发人员对数据库的使用不规范，查询的结果集过大，导致数据库的性能大幅下降；应用直接使用数据库的登录数据库，存在着一定的安全隐患。“数据安全访问平台”的目标是：建立数据统一访问接口及其使用标准，规范、约束和审计数据应用访问数据库的行为，对数据应用提供强制审计的技术手段。该项目具有较高的业务需求风险和技术风险。由于没有成熟系统做为参照，该项目需求不是很明确，而且系统涉及甲方多个利益相关方，各方对系统的安全和审计功能、运行维护、可靠性、性能和易用性有者不同的观点，某些观点之间还存在冲突。同时系统作为“数据中心”的基础设施之一，所有的应用系统都要通过本系统完成数据库访问。系统的可靠性和性能直接影响到应用系统的正常运行。整个系统分为 6 个子系统，包括 JDBC 驱动封装子系统、ADO.Net 驱动封装子系统、WebService. 接口子系统、管理配置网站、存储子系统(SQL Server2005 数据库，存储配置信息)和监控子系统（数据库网络协议分析与连接控制）。



無諾

摘要部分

- 1.时间、项目介绍、参与者的岗位、负责的事情、总体概括以及结果都有
- 2.虽然缺了项目规模和交付客户好评，但是因为整体通顺，问题不大



無諾



项目背景

- 1.项目产生的原因、项目的介绍、开发时间、功能的介绍、参与者的岗位和职责、项目所使用的技术都有
- 2.可以加上项目规模



無諾

过渡段

- 1.引出正文要讲的 4 个图，这里不建议写 4 个，3 个其实就够了，部署图可以不用讲
- 2.最后少了一个过渡句，过度到下个自然段进行项目介绍这几个图的内容

UML 在需求工作中的应用项目组采用一个精简 RUP 开发模型指导项目的整个开发流程。在初始阶段主要采用用户访谈、用户调查和联合讨论会捕获用户需求，进行初步分析，完成《需求规格说明书》初稿，通过用户评审；在细化阶段，对需求进行了细化，并在采用原型法验证用户需求，完成《需求规则说明书》更新稿，通过用户评审，作为项目验收的依据。项目开发中，我们采用了统一建模语言(UML)，并使用了 Rational Rose 工具。在需求工作中，我们主要使用了 UML 中的用例图、类图、活动图和部署图。

用例技术的应用整个需求开发都是围绕着用例技术开展的。首先，我们明确了系统的利益(查书)相关方、确定了系统边界和建设目标，以及主要功能特性。由于系统涉及甲方多个利益相关方，包括：应用科、维护科、网络科、信息中心领导和应用开发方，他们对系统的安全和审计功能、运行、维护、可靠性、性能和易用性有者不同的观点，同时本系统与已建成的网管系统和单点登录系统存在着交互关系，这给确定系统的目标和边界带来一定的难度。项目初始阶段开始，我们先通过用户访谈、用户调查了解了各个用户对系统的观点；随后，我们召开了联合讨论会，会上我们描述了各个用户的观点，以及之间可能的冲突，供各方进行充分的讨论，会议最后，信息中心的领导统一了各方的意见，对系统达成一致的目标，并明确了系统主要的功能特性，确定本系统与网关系统和单点登录系统的关系。其次，我们建立用例模型，并细化关键用例。明确了系统的目标和主要功能特性后，我们采用用例模型对需求进行分析。整个系统包括六个用例包：访问接口包用例包、审计信息管理用例包、用户认证信息管理用例包、数据库连接资源管理用例包、访问控制信息管理用例包、数据库连接监控管理用例包和审计日志管理用例包，共包含 55 个用例。对大部分用例我们只描述了一个基本正确流程，只对 5 个关键高风险用例进行了细化描述，包括：前置条件、后置条件、基本流、扩展流和相关约束。这项工作完成后，我们编写了《需求规格说明书》初稿，提交用户进行了审核。经过修改后，通过用户评审，作为初始化阶段的主要成果。最后，细化用例模型。在细化阶段，我们细化描述了所有的用例，完成《需求规则说明书》更新稿，通过用户评审后，作为项目验收的依据。

类图的应用用例技术描述了系统需求的动态结构，但对于需求特性和用例中出现的概念，并没有统一的分析。我们使用类图描述系统的核心概念。本项目中涉及的核心概念主要包括：逻辑连接、用户、应用、受限角色、认证 SQL 语句、参数取值范围的约束和查询结果集的限制。在分析核心概念时，我们主要关注：概念之间的关联关系，是否具有相同的生命周期，和具体的对应关系(一对多、多对多和多对一)。

部署图与活动图的应用由于整个系统包含多个子系统，各个子系统部署在不同的节点，需要考虑用户的网络结构是否能支持。在需求阶段，我们也描述了整个系统的部署。其中监控子系统比较特殊，由于其是通过分析 Oracle 数据库的网络数据包进行工作的，因此监控子系统必须接入核心交换机的镜像端口。我们将部署图与网络科进行了确认。我们使用活动图描述系统的应用场景。由于本系统对应用系统的开发增加了一层管理，因此应用系统的开发方与信息中心存在一个交互流程：应用开发方首先使用本系统的开发版进行本地开发，并填写基本配置数据；然后，在用户的生产环境中部署应用，并提交基本配置数据；最后，信息中心的管理员对配置数据进行修改后，应用在生产环境中才能运行。我们使用流程图描述了整个过程。用通道表示应用开发方和信息中心，并描述了各个通道内的流程以及通道之间的交互。

由于没有成熟系统做为参照，该项目具有较高的需求风险。由于在整个开发过程中使用了 UML 的用例图、类图、部署图和活动图，这使得我们能从多个方面完整的把握需求，有效的保证到了需求工作的质量。本项目的需求工作中，我们也遇到的一些问题，主要是维护 Word 文档与模型的一致性。我们使用 RationalRose, 建模，使用 Word 写文档，通过截图的方式将模型插入文档，因此需要手工维护两者的一致性。这种方式较繁琐，容易出错。今后的工作中，我们准备使用 Rational 公司的 Soda 工具，自动将 Rose 中的模型插入到 Word 文档中。



無諾

论文主体

- 1.最后一段的部署图和活动图应该分开写，如果提前就感觉部署图没什么好写的就应该直接去掉
- 2.回应手题目和论文主体混在一起写了，不建议这样写
- 3.删除线那部分的话其实是没有任何作用的，我们在论点清晰且有思路的情况下，尽量不要在正文去写无关的废话
- 4.跟用例图那段相比，类图这段写的太少了，就是在摆概念，并没有结合项目进行具体的讲解，类图的书写完全可以通过下面来书写：定义概念类、识别类的关系、为类添加职责，结合这几个点和项目来书写会更好
- 5.类图的对应关系还写错了



無諾

结尾部分

- 1.项目上线及运行效果没有
- 2.客户评价没有



范文 5：论软件系统架构评估

对于软件系统，尤其是大规模的复杂软件系统来说，软件的系统架构对于确保最终系统的质量具有十分重要的意义，不恰当的系统架构将给项目开发带来高昂的代价和难以避免的灾难。对一个系统架构进行评估，是为了：分析现有架构存在的潜在风险，检验设计中提出的质量需求，在系统被构建之前分析现有系统架构对于系统质量的影响，提出系统架构的改进方案。架构评估是软件开发过程中的重要环节。请围绕“论软件系统架构评估”论题，依次从以下三个方面进行论述。

1. 概要叙述你所参与架构评估的软件系统，以及在评估过程中所担任的主要工作。
2. 分析软件系统架构评估中所普遍关注的质量属性有哪些？详细阐述每种质量属性的具体含义。
3. 详细说明你所参与的软件系统架构评估中，采用了哪种评估方法，具体实施过程和效果如何

2016 年 3 月，我公司承担了国家某安全中心漏洞挖掘系统的开发工作，我在该项目中承担系统架构设计师的职务，主要负责系统的架构设计。该项目的主要目的是依托大数据平台从互联网流量中挖掘未知漏洞。

本文以漏洞挖掘系统为例，论述了软件系统的架构评估。首先分析了软件架构评估所普遍关注的质量属性并阐述了其性能、可用性、可修改性和安全性的具体含义。整个系统采用了面向服务 SOA 的架构设计方法。在架构设计完成之后，对 SA 评估采用了基于场景的评估方式中的体系结构权衡分析方法 ATAM，并详细描述了其评估过程，项目评估小组经过对项目的风险点、敏感点和权衡点的讨论后生成了质量效应树。目前系统已稳定运行一年多，从而验证了该项目采用 ATAM 架构评估保证了系统的顺利完成。

随着互联网的快速发展，网络上出现的安全问题越来越多，从互联网发展至今，已经爆发了众多的网络攻击事件，如网络蠕虫病毒感染、主机被控制、数据库被非法访问、非法电子银行转账等等。针对这些安全问题，很有必要开发一种 web 漏洞的发现和利用技术。2016 年 3 月我公司承接了国家某安全中心漏洞挖掘系统的开发工作。该项目通过对互联网中的流量进行特征分析，从中提取出相关的攻击内容，并将这些内容存储到大数据平台，结合大数据分析技术，对攻击者进行跟踪分析，从而捕获出未知漏洞。通过这种漏洞挖掘技术可以极大的解决大数据，大流量背景下 web 攻击入侵，帮助用户做好“事中”的安全工作，协助安全厂商对互联网攻击进行针对性过滤。

系统在整体架构上采用了面向服务的架构 SOA。前端采用了 PHP 进行开发，后台流量分析工作采用运行性较高的 c 语言在 Linux 服务器上开发，流量包存储使用了企业磁盘阵列，数据存储采用了 mysql。通过将系统拆分为多个子模块，各个子模块的构建上用服务进行了封装，它们之间通过消息进行通信。经过对客户需求的分析，我将该系统拆分为流量捕获模块(负责从互联网中捕获流量)、pcap 文件存储模块(负责将互联网中的流量存储到大数据平台)、流量分析模块(负责对流量进行分析验证)、数据库模块(负责漏洞数据的存储)和 web 管理模块(负责下发漏洞规则和查看漏洞信息)。下面先介绍下软件架构评估的质量属性。



無諾

摘要

时间、项目介绍、参与者的岗位、负责的事情、总体概括以及结果都有



無諾



项目背景

- 1.写了项目出现的原因，项目开始的时间，项目的介绍，项目功能的介绍以及相关技术；
- 2.缺少个人的岗位及职责
- 3.字数太多了，近 600 字了
- 4.过度到子题目有一句话描述，这里是可以的，但是略显生硬，最好是有一个过程
- 5.如果你写字数不够，可以加上项目规模



無諾

回应子题目

虽然只写了 7 个质量属性，但是这样也是可以的，针对重要的 4 个质量属性进行详细的介绍，也是没问题的

架构评估是软件开发过程中的重要环节，在软件架构评估中的质量属性有：性能、可用性、可修改性、安全性、可测试性、可靠性和易用性等。其中前 4 个质量属性是质量效应树的重要组成部分。性能是指系统的响应能力，即经过多长时间对事件做出响应。可用性是指系统能够正常运行的比例，通过用两次故障之间的时间长度或出现故障时系统能够恢复的速度来表示。可修改性是指系统能以较高的性价比对系统做出变更的能力。安全性是指系统能够向合法用户提供服务，同时拒绝非授权用户使用或拒绝服务的能力。

常用的架构评估方法有：基于问卷调查的评估方式、基于场景的评估方式和基于度量的评估方式。基于问卷调查的评估方式是由多个评估专家通过调查问卷的方式回答问卷中的问题，对多个评估结果进行综合，最终得到最终结果。其评价的具有主观性，不适合本项目。基于度量的评估方式虽然评价比较客观，但是需要评估者对系统的架构有精确的了解，也不太适合本项目。而基于场景的评估要求评估者对系统中等了解，评价比较主观，故本项目采用了基于场景的评估方式。基于场景的评估方式又分为架构权衡分析法 ATAM，软件架构分析法 SAAM 和成本效益分析法 CBAM。本项目中根据不同质量属性使用了 ATAM 作为系统架构评估的方法。

在使用 ATAM 进行架构评估时，我们根据项目需要成立了项目评估小组。其主要成员包括：评估小组负责人、项目决策者、架构设计师、用户、开发人员、测试人员、系统部署人员等项目干系人。我在这里的身份是项目的评估小组负责人和首席架构师。架构的评估经历了描述和介绍阶段、调查和分析阶段、测试阶段和报告阶段四个阶段。下面我分别从这四个阶段进行介绍。

在描述和介绍阶段，由于项目评估成员有部分人员对 ATAM 并不熟悉，我首先介绍 ATAM 的方法。它是一种基于场景的软件架构评估方法，对系统的多个质量属性基于场景进行评估。通过该评估确认系统存在的风险，并检查各自的非功能性需求是否满足需求。客户也阐述了系统的目的和商业动机。项目是为了通过捕获互联网流量从而挖掘出有价值的漏洞信息。通过实时获取漏洞可以有效的展开防御，保证网站的安全性。客户关注系统的性能及系统能否获取高质量的漏洞信息。最后作为架构设计师的我描述了系统将要采用的 soa 架构，并将系统进行了拆分，并讲解了各个子模块的功能，初步决定系统服务端在 Linux 下使用 c 语言进行开发。

在调查分析阶段，不同的需求方基于各自的考虑都提出了各自的要求。其中客户方提出：系统的要保证其可靠性，特别是针对黑客 in.进行跟踪的时，系统发生故障必须在 1 分钟内恢复，此优先级最高。经过自动化分析，系统对漏洞的自动识别率必须达到 90%以上，此优先级较高。系统可以对规则模块实时进行修改，其修改工作必须在 1 人天完成，以便可以根据最新的规则进行漏洞捕获。系统要确保一定的安全性。安全分析人员提出：系统需要过滤大部分正常的流量，以减轻安全分析人员的分析难度。系统必须提取出有价值的高风险 ip，无效的流量跟踪将会带来产出的低下。开发人员提出为了保证系统的开发效率及系统修改性，可以进行并行开发。经过总结我们获得了系统的质量效应树如下（考试时回简要图）。

针对这些场景我们分析了项目开发过程中的风险点、敏感点和权衡点。经过分析，该项目中存在以下风险点：黑客的 ip 如果不能实时捕获，将会丢失重要漏洞信息;系统中对消息的处理如果超过 12 小时，将会产生大量的消息积压。敏感点有：用户的加密级别、漏洞规则的修改。权衡点有：改变漏洞规则的严格程度会提升漏洞的准确率，同时带来系统性能的下降。改变系统的加密级别对系统的安全性和性能都会产生影响。

在测试阶段：经过评估小组集体讨论，确定了不同场景的优先级如下：系统的可用性最高，性能其次，可修改性及安全性优先级较低。在保证系统可用性方面，在流量捕获部分使用双机热备技术，在两个捕获系统之间设置心跳，当一台捕获系统出问题，另一台捕获设备接管。在流量自动化分析部分，采用了集群部署技术，一台分析设备出问题，不会影响整个分析系统。在保证数据安全性方面，磁盘采用企业磁盘阵列 raids 机制。在用户数据安全性方面，采用了非对称加密及信息摘要技术。

最后形成了评估报告，经过对架构的评估，确定了系统的风险点、敏感点、权衡点和非风险点，最后以文档的形式表现。其包括的内容包括：架构分析方法文档、架构的不同场景及各自的优先级、质量效应树、风险点决策、非风险点决策及每次的评估会议记录。

该项目开发工作于 2016 年 8 月完工，系统上线后，我们的安全分析人员和客户使用该系统对互联网流量进行漏洞挖掘，一共产生了 150 种以上的 web 流量攻击流量特征和 5 个未知 web 漏洞。在国家某安全中心网研室的其他项目中起到了支撑作用，尤其是某变量覆盖漏洞、某文件写入漏洞，某 sal 注入漏洞在项目使用过程中取得了一定得效果，得到了好评。为开展互联网安全事件得防御、发现、预警和协调处置等工作提供了数据依据，更好的维护了国家公共互联网安全，保障基础信息网络和重要信息系统的安全运行。



無諾

论文主体

这里从一开始介绍架构评估分类以及他们的区别和特点，再到具体的选择，然后从 ATAM 的四个阶段结合自己项目这四个阶段的具体内容进行了描述，没问题



無諾

论文结尾

- 1.项目上线及运行效果
- 2.客户评价
- 3.因为字数够了，所以没写不足之处和解决方式，这样是可以的

范文 6：论微服务架构及其应用

微服务提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通。在微服务架构中，每个服务都是一个相对独立的个体，每个服务都可以选择适合于自身的技术来实现。每个服务的部署都是独立的，这样就可以更快地对特定部分的代码进行部署。

请围绕“论微服务架构及其应用”论题，依次从以下三个方面进行论述。

- 1、概要叙述你所参与管理或开发的软件项目，以及你在其中所承担的主要工作。
- 2、简要描述微服务优点。
- 3、具体阐述如何基于微服务架构进行软件设计实现的。

本人于 2019 年 3 月参与了某公司的核心项目“在线问诊互联网医院平台”，该平台以在线问诊为核心，分为用户信息管理、病历管理、推荐处方、药品配送等功能。本人在该项目中担任架构师职位，主要负责整体架构设计和技术选型。

本文以该项目为例，主要讨论微服务架构设计在项目中的具体应用。首先根据业务领域和技术实现不同，本着单一职责的原则将系统划分众多服务，解决了业务模块高度耦合的难题，其次拆分后的服务利用 NACOS 进行服务治理来解决服务不好管理，配置文件分散的问题，最后通过 Netflix Hystrix 实现服务的熔断降级。防止因某个服务异常从而导致雪崩效应，使整条服务链宕机的问题，整个系统历时 1 年半，于 2020 年九月上线，至今已稳定运行 1 年，深受用户好评。

如今信息化时代，信息技术的快速发展以及“互联网+”概念的大力推进，为医疗行业的改革提供了一个新的探索方向，在这样的市场时代背景下，各种数据呈现爆炸式的增长，在线问诊系统开发也开始逐渐流行，也逐渐出现利用计算机辅助诊断的案例，这种模式有效的实现线下患者分流，进一步减轻当前中国医疗健康系统的负担。

我司在医疗行业深耕十几载，有良好的医疗基础，又拿下了互联网医院牌照，未来发展有巨大潜能，我司旧系统经过两年时间运行也累计了大量医疗数据，为未来智慧医疗打下坚实的基础，该项目主要以患者问诊，医生接诊为主要功能，凭借问诊效率高，无论身在何处都能为患者提供便捷的优质医疗资源的优势让业务快速扩张，用户量飞速增加，从而也使需求更加快速变化，以前的单体项目显然已逐渐无法适应我司对软件的要求，微服务改造刻不容缓。本项目组全体成员共 41 人，我在项目中担任系统架构师职务，我主要负责整体架构设计、技术选型、服务拆分等工作。

传统单体架构与微服务架构相比，以往的单体项目所有的功能模块都在一个进程中运行，各模块的之间的耦合度过高，关系十分紧密，基本是单个模块出问题，可能导致整个项目崩溃，各功能模块的扩展性很差，基本对某个功能模块进行扩展，经过调研，我们的系统将采用微服务的架构来进行开发，微服务架构主要有以下特点，1.微服务把每个职责单一的功能放在一个独立的服务中。2.每个

服务允许在一个独立的进程中。3.每个服务有自己的数据存储，实际上，每个服务应该有自己独享的数据库、缓存、消息队列等资源。4.每个服务有多个实例在运行，每个实例可以运行在容器化平台内，达到平滑扩展伸缩的效果。5.每个服务应该有自己的运营平台，以及独享的运营人员，这包括技术运维和业务运营人员。每个服务都高度自治，内部的变化对外透明。6.每个服务都可以根据性能需求独立的进行水平伸缩。本文着重从微服务拆分，微服务治理，限流熔断在本项目中的具体应用效果来进行介绍。



無諾

摘要

时间、项目介绍、参与者的岗位、负责的事情、总体概括以及结果都有



無諾

项目背景

- 1.写了项目出现的原因，项目的介绍
- 2.可以加上项目的开始时间
- 3.可以加上项目所采用的技术，而且项目的介绍略显单薄
- 4.没有过度段，直接从下面开始写回应子主题有些生硬



無諾



一、微服务拆分

通过分析核心业务系统的需求，我们拆分服务秉承一个原则，就是不同微服务的边界明确，不重复开发相同的业务，也就是职责单一原则。并且我们对数据库也进行来拆分，要求每一个服务都有自己的数据库，不能直接访问其他服务的数据库。每个微服务可以将自己的业务暴露成服务，供其他服务进行调用。最终我们将核心的业务系统拆分为：网关服务、认证中心服务、权限服务、用户中心服务、IM 服务、支付服务作为基础服务层。诊疗服务、订单服务、处方服务、病例服务作为业务服务层。定时服务、文件服务、日志服务、分布式事务服务作为工具服务层。每个业务模块都是根据业务领域进行划分，同一个业务领域下因质量属性要求不同需要划分为两个服务，如认证服务和用户服务，由于认证服务是一个比较独立的模块，他只处理用户的登陆登出，作为应用的入口他的可用性和性能的质量属性要求相对较高。用户服务宕机不会影响认证服务。用户可以就登陆系统操作订单，问诊等功能，使我们的系统容错率更高。

二、微服务的治理

由于原单体项目被拆分为众多服务，那服务消费者如何获取服务提供者的具体调用地址，因此引入了 Nacos 注册中心，每一个服务启动时都需要在注册中心进行服务注册，注册中心会保存这些信息如 ip 和端口的信息，所以每次服务消费者想要调用某一个服务的时候都可以在注册中心，通过服务名查询到对应服务的 IP 端口地址，来进行服务访问，服务消费者可以通过负载均衡的算法，从同类型服务列表中选择一个服务来进行远程调用，同时我们要求服务提供者每隔 30 秒向注册中心发送心跳请求，来确认其健康状态。如果注册中心感知到服务异常就会将其剔除。保证就算有异常服务我们还是能快速发现使其下线，保证系统的可用性，同时对于我们这么多服务的配置文件，我们也采用了 nacos 的配置中心来进行管理，比较好的一点支持动态配置，在某些功能的切换上面，比如：验证码单日条数的限制，日志的开关，我们都不用让服务重新发版，这也降低了在修改代码和发版过程中人为造成错误的风险。

三、限流熔断

在我们需求分析中，发现诊疗服务，依赖于 IM 服务、处方服务、订单服务，而处方服务和订单服务又依赖于其他服务，这样下去调用链路很长，也是我们的核心业务，当诊疗服务的链路上有几个调用的子服务宕机或者延迟较高的话，会导致请求被堵住，堵住的请求会消耗掉服务器的资源，当这些请求越来越多，服务来不及处理，慢慢的会导致其他的请求同样无法处理，最后导致整个业务系统卡顿乃至崩溃，调研发现目前最常用的服务依赖保护是熔断和限流，我们通过 Hystrix 实现了断路器模式，当每个服务节点发生故障之后，通过断路器的故障监控，向调用方，返回一个符合我们预期的备选响应，还不是长时间的等待服务响应，这样保证了服务在整个调用链不会长时间的等待响应，不必要的占用资源，防止故障在整个微服务系统中蔓延，最后乃至雪崩的情况发生。

回应子主题

- 1.通过对比传统架构和微服务架构来引出微服务的优点，这样写是可以的
- 2.没有必要举出那么多个例子，哪怕有，我们也需要找到最典型的 2,3 个进行详细点的介绍，不建议这样所有都泛泛而谈
- 3.最后一句话引出正文，这样是可以的



無諾

论文正文

它这里是从项目改造的角度来分为 3 个论点来书写，我们不一定需要按照他这样写，我们完全可以写新开发一个项目，然后基于微服务的优点，我们选择了微服务的架构，然后从几个优点的角度作为论点来书写

项目在 2020 年 9 月份通过了最终验收，上线，至今以来系统运行也非常稳定，已经连续运行一年多，已有 4W 以上的医生在平台注册，百万以上的患者得到服务，获得来客户一致的好评，在系统管理上每个服务都有专门的人来管理，提升了产品的体验，也提升了服务客户的质量和效率。同时每次发版也不会对整个系统发版，而是对某个修改的服务发版，面对需求也能快速响应，相比于之前的单体项目，系统逻辑更加清晰，面对日益增加的用户量我们也能通过各个服务的水平扩容来提升系统的负载。但是微服务改造之后，产生了众多服务，运维的开销及成本增加，后续会通过搭建 devops 平台，运维人员需要写好自动化运维脚本，通过自动化工具可以实现自动发布和监控，可以降低维护成本，提升上线交付效率和质量。

实践证明系统平稳运行与采用来合适的架构风格密不可分，经过此次微服务架构的应用方法，也看到不足之后，在未来还会不断更新知识，完善本系统的架构设计，使整个系统更加合理，更有效的服务于医生和患者。



無諾

论文结尾

- 1.项目上线及运行效果
- 2.客户评价
- 3.没有展开写不足之处和解决措施，而是写了项目改成微服务之后的一些好处，这样写也是可以的，但是还是没有写不足之处和解决措施好

2022 年 4 月，我参与了某市水务集团的智慧水务管理系统项目的建设工 作，并担任了系统架构师一职，负责该系统的架构设计和技术规划。该项目致力于打造一个集表务、营收、报表等功能为一体的综合性管理系统，旨在全方位提升水务集团的日常工作效率。

本文以该智慧水务管理系统为例，主要讨论了微服务架构在项目中的应用。微服务架构将单一系统拆分为多个独立服务，每个服务可根据实际业务需求选择合适的技术栈，并能独立开发、测试和部署。这种架构确保了单个服务的故障不会影响到整个系统，具有单体架构不具备的优势。因此，我选择了微服务架构作为系统的基石。该项目历时 10 个月，于 2023 年 2 月正式上线，并平稳运行至今，赢得了客户的高度赞誉。

在国家政策的积极引导和支持下，水务行业的数字化转型已经是行业发展的必然选择。在数字化转型前，水务行业普遍面临着运营效率低下、客户服务响应慢等问题，已经无法满足现代化的需求。因此某市水务集团积极响应国家的号召，委托我公司承建智慧水务管理系统，以进行数字化转型。

本项目自 2022 年 4 月开始，我在项目中担任了系统架构师一职，主要负责架构设计、技术选型、架构实现等工作。该系统包含了表务模块、营收模块、报表模块等功能模块，具体而言，表务模块主要负责水表档案管理以及抄表数据采集，并进行水费计算；营收模块专注于用户账户管理、水费收费流程以及发票开具；报表模块则承担抄表、收费等数据的统计分析工作。该系统历时 10 个月，于 2023 年 2 月正式上线，并平稳运行至今，为客户的日常工 作提供了全方位的软件支持，解决了客户之前工作上的痛点和难点，赢得了客户的高度赞誉。由于在本系统中使用了微服务架构，所以系统得以能够按时且高质量的完成。

微服务架构相较于单体架构，其特点体现在复杂应用的解耦、服务的独立性、技术栈的灵活选择、系统的易扩展性和出色的容错能力等方面。通过将复杂的单体应用拆分为多个专注于特定业务需求的小型服务，微服务架构实现了复杂应用的解耦。这些服务能够独立进行开发、测试及部署，服务的独立性显著提升了开发效率。同时，每个服务都能根据业务需求自由选择合适的技术栈，进一步增强了系统的灵活性和适应性。此外，微服务架构能够根据实际需求对每个服务进行独立扩展，确保系统在面对突发流量时能够迅速响应，保持高可用状态。更重要的是，即使某个服务发生故障，也不会导致整个系统的瘫痪，提高了系统的容错能力。因此，我选择了微服务架构作为系统的基石，接下来结合项目实际情况探讨微服务架构的应用。

H3C
开头没问题



知乎知识堂

H3C

这一段也没问题，但是我提个建议，你可以在讲述微服务架构之前，先用一句话描述下你们这个系统的某个痛点，而这个痛点正好可以用微服务架构来解决，然后再展开说下微服务架构，这样会更好，比如说该水务管理系统在 xxx 方面要求比较高，而微服务的某个特点正好可以解决这个问题，所以选择了微服务架构，再对该架构进行一个简单说明

H3C

前面都写的不错，最后这句话应该是过度有点问题，是否使用微服务架构跟你系统是否能按时完成是没有必然关系的，而且这句话应该是要引出下面那段对第二个问题的回答，从你下面那段来看，你是讲述微服务的特点，所以这句过渡句你可以写成：由于该系统选用了微服务架构，所以极大增强了系统的 xxx、xxx、xxx 等特性，下面针对微服务的这些特点进行介绍，这样的话，引出下面这段就更好了



复杂应用的解耦势必会出现服务管理的问题。为了解决这个问题，我采用了 Spring Cloud Alibaba 微服务框架，这一框架具有强大的功能和良好的生态系统。在此基础上，我结合 Nacos 进行服务的注册、发现以及配置管理，这一组合极大的简化了服务的管理流程，提升了服务的稳定性和可维护性。为了进一步提升服务间的调用效率，我选择了 Dubbo 作为 RPC 远程服务调用框架，Dubbo 具有高性能和透明化的特点，能够很好地解决服务间高效调用的问题。由于服务被拆分成多个，当某些操作需要同时调用多个服务时可能会出现部分服务操作成功而部分服务操作失败，进而导致数据不一致的问题。针对这一状况，我引入了 Seate 来进行分布式事物的一致性管理。通过这一措施，能够确保各个服务之间的操作要么全部成功，要么全部失败，从而有效防止了数据不一致现象的发生。此外，我还使用了 Gateway 作为 API 网关，对所有前端的请求进行统一管理和路由，根据负载均衡策略调用同一服务的不同实例，起到了资源合理分配以及快速响应的作用。↵

服务的独立性为技术栈的灵活选择及出色的容错能力奠定了坚实基础。这种独立性体现在每个服务的高度自治上，它们能够自主进行开发、测试及部署，且各服务间的进度互不干扰。得益于高度自治，各服务可以根据实际需求，自由选择最适合的技术栈。例如，表务服务需要在每天凌晨接受大量抄表数据上报，便采用 RabbitMQ 信息队列进行抄表数据的分发和处理，有效减轻系统的负担，提升稳定性；而营收服务涉及退款、余额调整等复杂流程，

则引入了 Flowable 工作流引擎进行流程的设计和管理，增强了流程的可维护性。服务的独立性有效降低了系统耦合度，即使某个服务出现了故障或暂时不可用，其他服务仍能正常运行，确保系统整体不因单一服务失效而奔溃，显著提升了系统的容错性。↵

H

H3C

这段写的不错↵

H

H3C

引出问题，解决问题，配上技术和流程，就该这样写↵



系统的易扩展性至关重要，它要求在面对海量且突发的流量时，系统必须能够自动执行一系列操作，如限流、扩容和降级等，以确保其稳定性；当业务流量回落时，系统需要智能地回收多余资源，从而避免浪费。容器化技术以其高度的服务独立性和可移植性，成为了在不同环境中轻松部署和运行服务的理想选择。因此，我采用了 Docker 容器化技术，对每个服务及其运行环境进行统一封装。同时，结合 k8s 进行容器集群的管理和编排，实现了根据实际业务需求自动调整业务服务规模的能力，既确保了系统的高可用性，又实现了资源的合理利用。鉴于服务容器数量庞大，手动部署已不再是合理的选择。为此，我引入了 Jenkins 进行代码构建、测试和部署。通过 k8s 和 Jenkins 的紧密结合，实现了系统的自动化部署，显著提高了部署效率。此外，在每个服务中，我使用了 Flyway 来自动执行因变更而需要执行的 SQL 语句，从而避免了人工操作数据库可能带来的失误和系统不可用风险。同时，我还采用了 Sentinel 对流量进行监控，一旦流量出现异常，Sentinel 能够自动采取限流或降级措施，有效防止服务因资源耗尽而引发的雪崩效应，进一步保障了系统的可用性。

本系统自 2023 年 2 月正式投入运行以来，始终保持平稳运行，这一成就于微服务架构的选用息息相关。在项目运行过程中，尽管遭遇了数次突发流量高峰的调整，但得益于微服务架构的易扩展性，系统能够迅速进行资源扩容，从而确保在流量激增的情况下依然能够稳定的提供服务。本系统在水务集团的日常工作中展现出了显著的优势，极大地提升了工作效率。它不仅助力水务集团在数字化转型的道路上迈出了坚实的一步，还赢得了水务集团的高度评价和嘉奖。

通过本项目系统架构的设计和实现过程，我深刻认识到微服务架构对于提升系统的容错能力、提高系统的可扩展性以及保障系统可用性的重要作用。这一经历让我更加清醒的认识到，在软件系统架构的这条道路上，我仍有大量知识需要学习，还有很长的路要走。然而，正如屈原所言：“路漫漫其修远兮，吾将上下而求索。”我将以坚定的决心和不懈的努力，持续探索、学习和实践，不断提升自己的专业素养，致力于构架更加卓越的软件系统架构。

H

H3C

后面这两段也写的挺好。

H

H3C

结尾也写的不错，整体来说，这篇论文是一篇非常优秀的论文了，我给分的话，可以打 62-65 分了，虽然有一些小问题，但是瑕不掩瑜，考试按照这种方式写，必过。

范文 7：论云原生架构及其应用

论云原生架构及其应用（AI 生成）

云原生架构（**Cloud-Native Architecture**）是一种基于云计算特性设计的系统架构范式，旨在充分利用云服务的弹性、可扩展性和高可用性。其核心技术包括容器化（如 **Docker**）、微服务、持续交付（**CI/CD**）、服务网格（如 **Istio**）和声明式 **API** 管理等，能够显著提升资源利用率、加速业务迭代并增强系统容错能力。然而，云原生架构的实施也面临多云适配、安全治理、监控运维等挑战。请围绕“云原生架构及其应用”论题，依次从以下三个方面进行论述：

- 1.概要叙述你参与管理和开发的云原生架构项目，并说明项目的核心目标与业务驱动力。
- 2.简要分析云原生架构的核心特征与技术组成（如容器编排、微服务治理），并对比其与传统单体架构、**SOA** 架构的异同及适用场景。
- 3.具体阐述你在项目中如何设计并落地云原生架构，包括技术选型（如 **Kubernetes** 集群设计）、关键问题（如跨云部署、冷启动延迟）的解决方案，并说明实际应用效果与优化方向。



2019 年 3 月，我单位联合某高校研发了《程序在线评测比赛考试系统》。系统以程序代码在线提交自动评测功能为核心，分为题库模块、评测机模块、实验作业模块、考试模块、比赛模块、抄袭判定模块、用户管理模块等，支持对接教务平台。在项目中我担任系统架构师，负责架构设计工作。

本文以该系统为例，主要论述了云原生架构在项目中的具体应用。系统以 [SpringCloud](#) 微服务框架开发，分为前端 **Web** 服务、平台保障服务、业务服务三部分。前端 **web** 服务由负载均衡与服务器集群结合，实现高并发的前台界面；平台保障服务以 **Eureka** 为中心，由 **API** 网关、服务注册中心、监控平台等构成，实现基础服务框架；业务服务划分为多个微服务，基于 **Docker** 容器，协同工作实现具体业务功能。最终系统顺利上线，获得用户一致好评。

笔者在一个专为高校建设计算机专业智能教学一体化平台的单位任职，过往成果有《计算机组成原理仿真实验系统》等。2019 年 3 月，我单位联合某大学研发了《程序在线评测比赛考试系统》项目(以下简称为“OJ 系统”)，以取代原有传统的编程上机考试平台。

系统以程序代码的在线提交自动评测功能为核心，主要分为题库模块、评测机模块、实验作业模块、考试模块、比赛模块、抄袭判定模块、用户管理模块等。题库模块主要负责试题和测试用例的管理，用户根据试题要求编写程序代码提交到系统，系统将测试用例与程序代码发送给评测机模块，由评测机自动编译、执行、判分，并将结果发送给其他相关模块进行统计；实验作业模块用于在线布置作业，从题库中选取试题，设置截止日期等要求；考试模块用于学生在线考试，按教师预先设置的参数自动从题库随机抽题生成试卷，以及向教务平台上传考试成绩；比赛模块主要用于 **ACM** 竞赛的培训；抄袭判定模块用于鉴定代码与他人代码雷同率；用户管理模块负责用户信息的管理。在这个项目中，我担任了系统架构师的职务，主要负责系统的架构设计相关工作。

云原生架构以微服务和容器技术为代表，有服务化、强韧性、可观测性和自动化四类设计原则。通过服务化的设计原则，应用被分解为多个服务，可分别选择不同的技术，单个服务模块很容易开发、理解和维护，无需协调其他服务对本服务的影响；通过强韧性的设计原则，微服务可以分布式云化部署，负载均衡管理请求的分发，避免单机失败对整体服务的影响，以及弹性调整资源容量；通过可观测性的设计原则，能够对系统进行健康检查、指标监控、日志管理和链路追踪，提高系统运维、管理和排错能力；通过自动化的设计原则，可实现系统的自动化部署、自动化扩展伸缩、自动化运维、持续交付和集成，有效减少人工操作的工作量。



H3C

摘要

时间、项目介绍、参与者的岗位、负责的事情、总体概括以及结果都有



H3C

这句话没有必要，如果想介绍自己以往成果或者说是这个行业的经验的话，这句话重量不够，若没有拿得出手的经验还不如不写



H3C

项目背景

- 1.项目诞生原因在上面已经写到了，可以加上取代之前系统的原因。
- 2.这里写的是背景中的功能介绍，但是不需要写那么多模块出来，找出 3-4 个模块跟摘要中的模块介绍进行呼应介绍即可，这里写的冗余了
- 3.这段的最后介绍略显单薄，而且缺少引入到下一段的过度语句



H3C



回应子主题

根据这段的描写,子主题应该是需要阐述出云原生架构的设计原则,这里提取出 4 个原则进行解释是没问题的



H3C

子主题到正文的过渡段很有必要

0J 系统基于 Spring Cloud 微服务框架开发,将平台服务划分为三类,分别为前端 Web 服务、平台保障服务、业务服务。下面针对这三类服务展开具体说明。

1.前端 Web 服务

前端 Web 服务主要提供给用户使用的界面,分为前置 Nginx 负载均衡服务器、前端网站 Nginx 集群。当用户通过网络访问系统时,首先会访问到前置的 Nginx 负载均衡服务器,负载均衡服务器会将请求转发到前端网站的 Nginx 集群,前端网站通过发起 http 请求来和后端交互,具体是通过 Ajax 方式来调用后端 RESTAPI 接口。用户访问网站通过前置的 Nginx 负载均衡服务器来转发到前端网站集群,以起到将用户请求进行分流的作用。当前端网站集群中的部分服务发生故障时,系统仍可正常地对外提供服务。前置 Nginx 负载均衡服务器使用软件反向代理的方式来实现负载均衡,部署为路由模式,系统内部网络与外部网络分属于不同的逻辑网络,以实现系统内部与外部网络的隔离。在负载均衡算法的选择上,使用最小连接法,每当用户的请求来临时,任务分发单元会将任务平滑分配给最小连接数的前端网站节点,这样的架构以廉价且透明的方式扩展了服务器和网络的带宽,可以大大提升系统的并发量,同时保证网站前端整体的稳定性和可靠性。

2.平台保障服务

平台保障服务用以实现后端微服务的基础框架,包括 API 路由网关、服务注册中心、服务监控组件。API 网关收到前端的请求,不会直接调用后端的业务服务,而是首先会从服务注册中心根据当前请求来获取对应的服务配置,随后通过服务配置再调用已注册的服务。当后端微服务存在多个实例时,将采取负载均衡的方式调用。服务注册中心是整个云原生架构体系的核心部分,由 SpringCloud 的 Eureka 组件来实现,专门提供微服务的服务注册和发现功能,涉及三种角色:服务提供者、服务消费者和服务注册中心。API 路由网关、所有业务服务,以及服务监控平台组件都注册到服务注册中心。面过服务注册中心两两互相注册、API 路由网关向服务注册中心注册多个实例等方式,来实现后端整本服务的高可靠性。服务监控平台通过注册到服务注册中心,获取所有注册到服务注册中心的后端业务服务,从而监控到所有后端业务服务的运行状态信息,最后收集并展示整个微服务系统的运行状态,更进一步保证整个后端的服务质量。



3.业务服务

业务服务按功能模块，相应划分为题库服务、评测机服务、考试服务、比赛服务、抄袭判定服务等。各服务单独打包，基于 **Docker** 容器，连同运行环境一起封装，根据实际情况可在一台或多台物理机同时部署多个实例，服务启动后将自身信息注册到服务注册中心。服务间协同工作，通过松耦合的服务发现机制，动态调用对方 **RESTAPI** 接口。对于压力较大的服务，如评测机服务、抄袭判定服务等，将部署为多实例集群。以在线考试功能为例，用户进入考试时，考试服务核验考生信息通过，周周题库服务，题库服务返回试题，由考试服务组合成试卷，返回前端显示。用户交卷时，提交的程序代码到达考试服务，考试服务拆分后分发给题库服务，题库服务将程序代码和测试用例送入 **MQ** 队列排队。然后由负载均衡机制，依次将队列中待评测程序分发给评测机服务编译、执行、判分，完成评测后，题库服务统计试题通过率，考试服务统计成绩并向前端显示。在此期间服务请求者无需了解其他服务对数据如何具体处理和分析。

系统自 **2019 年 10 月** 正式上线已运行一年有余，在学校的日常教学考试和竞赛培训中投入使用，至今已有 **3000** 名以上的学生用户，评测了 **70000** 份以上的程序代码，获得了单位同事领导和学校教师们的一致好评。在开发和试运行过程中，主要遇到了两个问题。一是跨域问题。**OJ** 系统前后端分离，前端通过 **Ajax** 访问后端服务。由于浏览器同源策略的限制，导致前端 **UI** 无法正常访问不同端口和 **IP** 的后端服务。我们利用 **Spring Boot** 后端的 **Cors** 跨域机制解决了该问题。二是评测机宕机问题。评测机服务需要执行用户提交的代码，部分用户短时间内提交了大量不安全代码，导致评测机集群中所有实例全部宕机。我们引入心跳机制、快照回滚机制，以及基于机器学习技术的预判断机制，使评测机宕机时能够在 **10 秒** 内自动重置恢复运行，最终解决了该问题。

实践证明，**OJ** 系统项目能够顺利上线，并且稳定运行，与系统采用了合适的架构设计密不可分。经过这次云原生架构的方法和实施的效果后，我也看到了自己身上的不足之处，在未来还会不断更新知识，完善本系统在各方面的设计，使整个 **OJ** 系统能够更加好用，更有效地服务于高校师生。



H3C

正文

- 1.正文写的很正规，每一段开头先对这个服务进行简单阐述，然后再进行具体讲解，这样不错
- 2.每一段都会涉及到多项技术实现，我们也需要这样写
- 3.第一段和第三段都有涉及到一些场景的解决方案，这样是加分项
- 4.整个过程几乎没有涉及到云原生架构的内容，也没有突出使用云原生之后的好处，这是最大的败笔，



H3C

结尾

在结尾处写了项目中所存在的问题，但是第一个问题实在太低级了，再说了，这个问题是应该由架构师考虑的吗



H3C

最后记得呼应论文题目点题



范文：论集成云服务系统在云平台中的设计与应用



论集成云服务系统在云平台中的设计与应用（AI生成）

随着企业数字化转型深入，企业内部HR、OA、财务、制造、税务等业务系统独立建设，普遍存在数据孤岛、异构系统难以互通、业务流程无法协同等问题。云平台集成服务系统可通过统一连接器、流程编排、事件驱动、数据集成能力，实现多异构系统的数据互通与业务协同，是企业数字化中台建设的核心基座。在大型SaaS云平台建设中，如何通过合理的架构设计，解决异构适配、数据一致性、多消费者竞争、流量管控等核心问题，成为架构设计的关键。

请围绕“云平台集成云服务系统的架构设计与应用”论题，结合你参与的实际项目，依次完成以下论述：

- 1、简要叙述你参与的云平台集成服务项目以及你承担的核心架构设计工作。
- 2、简述云平台集成系统分层插件架构的核心思想，说明平台基础设施层、消息适配层、连接器扩展层、核心业务层等分层架构的主要职责与协作关系。
- 3、结合项目实践，详细论述你在项目中解决异构系统集成的核心设计方案，重点说明跨系统数据一致性保障、精细化流量治理、分布式多消费者竞争问题的实现思路、技术选型与落地过程。

论集成云服务系统在云平台中的设计与应用

2022 年 3 月至 2023 年 6 月，我参与了某公司产品“集成云服务平台”（以下简称“集成云”）项目建设。该项目是我公司 SaaS 产品和云平台的核心组成部分，服务于数千家企业客户，日均处理数据同步任务超过 10 万次。集成云不但承担着需要为原 ERP 用户逐步升级到 SaaS 云订阅服务的核心任务，同时也需要为企业或集团内外部多个不同异构系统提供进行数据集成的核心要求。项目研发团队超 20 人，项目总投资超 1000 万元，作为平台长期研发项目，一期二期总历时超过 12 个月，我作为核心技术架构之一，负责数据复制和流程编排引擎，及连接器模块的架构设计与实现。项目一期于 2022 年下半年成功上线，并且不断迭代与维护，作为公司云平台的基座组成部分之一，获得众多企业客户和伙伴的良好评价，并成为公司质量金奖项目。本文以该项目为例，论述集成云在微服务架构下的数据协同，流程引擎和事件驱动的核心设计与落地实践。

现代企业的数字化与信息化建设开始逐步走向企业平台管理，随着企业业务和规模的快速扩大，不同业务线和部门之间会创建不同的信息系统，如 HR，公司 OA，制造云，税务云，财务管理等，多个系统之间不能进行数据流通，各自系统数据和服务单独治理形成各自的信息孤岛，也无法形成统一的整体决策和管理。我公司是国内企业级管理软件知名供应商，旨在为客户建立完善的企业管理平台，不仅仅是为企业提供多个业务线上的管理软件产品，同时更要为所有企业内外部系统建立一个统一的企业数字化平台，集成云就是其中作为黏连所有系统的数据通路的至关重要的一个部分。



WPS_266548598

时间、项目介绍、团队介绍、项目金额、岗位、职责，但是最后一句的内容跟后面的论点



WPS_266548598

项目的背景描述



2022 年初，平台决定建设集成云，为我公司云平台下各个云服务产品之间建设数据通路服务，同时也需要和客户公司内其他异构系统形成数据协同。让我公司云平台产品成为企业数字化管理的核心基座。集成云是建立在云平台的微服务架构基础之上的独立应用，所以集成云引入了”基于云平台下的分层插件架构”，建立五层分层架构，包括平台基础设施层、消息队列适配层、连接器扩展层、核心业务层、接入和表现层。其中自研的平台基础设施层为我们的整体架构提供了服务注册，缓存，数据库，动态领域模型管理等所有基础设施。我作为核心架构之一，主导连接器扩展层、消息队列适配层和部分核心业务层的设计与实施工作。重点解决异构系统集成数据一致性保证、流量精细化管理、事件多消费者竞争等核心问题。

在项目实施前中期，我深入研究了集成服务的核心组件和协同机制，其核心设计思想是通过连接契约屏蔽异构，以配置驱动替代编码，以事件编排解耦流程，从而通过产品配置统一解决企业系统集成与数据集成。

集成云主要包括连接器和元数据，集成管理，服务编排三大部分组成，云平台会为所有活动建立监控统计和日志追踪机制。连接器和元数据负责管理所有的内外部连接器配置及来源元数据，通过预先同步外部系统的结构元数据（包括服务接口，数据表，实体模型等），将异构系统的能力描述统一存储在集成云服务内部。实现集成配置的可视化，可验证和低错误率。集成管理面向数据，基于 ETL 管道模型，使用 QuartzCronExpressin 解析定时规则或事件驱动，并通过 Redis 分布式锁保障集群环境下任务互斥执行，他包括数据集成，API 集成，消息集成等多种方案，管理完整的集成方案包的全生命周期。服务编排面向流程实例，通过整合已有的连接器和元数据，通过动作和多步骤编排，彻底解决跨系统的多步骤集成方案问题。



WPS_266548598

回应第一个子问题，并对项目进行详细一些的介绍，有涉及到业务和技术，建议加上一些具体的技术名词



WPS_266548598

这一段按理来说应该是一个过渡段，引入第二个子问题，并且把纯概念性的回答引入到自己的项目业务中去，但是我没看到这一段和下一段的必然关系



WPS_266548598

这一段是回应第二个子问题，根据回答的内容可以看出子问题估计是：集成云有哪些组成部分，这些组成部分的作用是什么，这里对这些子问题的回答缺少跟自己的项目结合



基于上述设计方案,我将重点围绕集成管理和服务编排共同解决企业集成场景下的数据一致性问题,连接器模块深度配置实现流量精细化管理,管理集成方案运行时的 Redis 分布式锁控制多消费者竞争问题三个维度开展工作。

企业集成场景下的数据一致性问题区别于单系统数据库和事务的数据一致性问题,核心挑战是跨系统层面数据一致性问题,由于跨系统无法共享,系统之间调用和业务操作不可逆,网络等原因可能引起系统数据集成失败。我处理的核心思路遵从:用最终一致性替代强一致性,用补偿替代回滚。我构建了 E (Extract 提取) -T (Transform 转换) -L (Load 加载) 三位一体的可观测基础,每一个阶段都存在自己的基于 AtomicLong 的独立计数器,三段计数器把“任务失败”这个模糊的状态,分解成了哪个阶段,失败多少,失败原因的精确信息。建立失败检测机制的观测保障后,在转换阶段和加载阶段采取多种策略保证集成数据一致性,候选键 + 幂等写入保证批次写入的原子性和最终写入一致性;分片 + 断点续传执行批次内多分片并行执行,每个分片独立执行单元有自己组成的独立执行状态,当执行单元发生失败,通过执行重试;指数退避重试保证在任务失败后,采取立方数策略,解决任务瞬时抖动,目标系统瞬时故障等常见原因引起的执行失败,确保重试后的保证执行。

在连接器扩展层,我们通过接口契约把连接配置和连接运行时彻底分离,来屏蔽系统异构性对整个核心集成逻辑引擎的影响,这是一个多对多问题,通过建立统一扩展层,利用策略模式完全隔离内部引擎逻辑和外部连接器配置的耦合。在运行时连接层配置通过构建目标系统的 License 控制准入策略,引入 HikariCP 管理连接数上限控制,设定任务调度线程,利用令牌桶思想管控调用频率、限流、并设定熔断要求,当目标系统故障失败达到阈值时自动停止任务。

由于云平台是一个分布式微服务架构,当完成连接器配置被用于执行数据复制任务时,任务的调度和执行过程中对于多节点集群部署,就会产生多个消费者竞争执行的问题,在这



WPS_266548598

过渡段,下面引出论点,但是这里你要围绕的三个维度不是很容易看出来,因为维度太长了,建议考试的时候加粗显示



WPS_266548598

建议先对论点进行解释,再结合项目进行论述



WPS_266548598

论点的叙述方式是:论点+论点的解释+背景+方案提出+基于业务进行选型+涉及到的技术+实行结果,按照这里来看,这里的论点叙述有背景+方案+实行结果,缺少了涉及的技术以及对过程描述的过于详细



WPS_266548598

一样缺少对论点的解释



由于云平台是一个分布式微服务架构，当完成连接器配置被用于执行数据复制任务时，任务的调度和执行过程中对于多节点集群部署，就会产生多个消费者竞争执行的问题，在这里，我们引入了 Redis 和内部 Rabbit MQ 消息集成来解决问题，虽然使用的是平台提供的自研框架 dlock，但是原理选型上，依然是标准的 Redis 分布式锁实现。通过调用设定 Lua 脚本设计实现了方案级互斥锁和数据级互斥锁，共同作用解决任务队列竞争问题，并通过 RabbitMQ 管理大批量任务入队问题进行高并发状态下的削峰处理，实现任务异步化处理，保证消息只被一个消费者取走。

集成云一期于 2022 年 8 月上线，后不断迭代版本，在 2023 年中完全稳定面对内外部各种集成任务的挑战。于 2023 年中，稳定周期内，为各大企业客户的数据集成任务提供了大量集成标准方案，从原来的专业开发集成数据脚本或程序方案，通过配置集成云方案包，把一个系统的集成工作量大幅度从月周期降低到 2-5 日周期范围内，获得公司内外和客户的一致好评，并获得当年公司内质量金奖项目。

通过该项目，我深刻认识到对于架构过程中，引擎稳定和内外部配置模式的插件化设计思想的作用，并且如何利用平台思维搭建业务服务架构，集成云本身属于单体插件化的应用架构，虽然运行在微服务的架构平台之上，但是利用业务差异化的平台能力，恰好解决了具体的场景问题也是一种成功的架构方法论。



WPS_266548598

一样缺少对论点的解释



WPS_266548598

这一段还行，有矛盾点，有解决方案，有技术，有结果



WPS_266548598

项目统一的结尾，需要背下来



WPS_266548598

进行技术或者业务的总结，没问题



论高并发系统的架构设计与性能优化

随着互联网业务的快速发展与用户规模的持续增长，高并发场景已成为企业信息系统面临的核心挑战之一。大促活动、热点事件、爆款商品抢购等场景往往会在短时间内产生千万级甚至亿级的瞬时流量洪峰，对系统造成巨大的准入压力。传统的系统架构受限于纵向扩展的瓶颈、数据库连接池上限以及同步阻塞的I/O模型，极易在流量高峰时出现响应超时、资源耗尽、服务雪崩，甚至全站不可用等问题。如何通过合理的架构设计，在确保数据最终一致性的前提下有效承接瞬时高并发流量，同时保障核心链路的稳定运行，是衡量架构师设计能力的关键标尺。

请围绕“论高并发系统的架构设计与性能优化”论题，依次从以下三个方面进行论述：

1. 概要叙述你参与管理和开发的软件项目以及你在其中所担任的主要工作。
2. 论述高并发系统的核心设计原则与关键技术手段（如弹性伸缩、缓存、异步、限流、降级等），并说明这些技术如何协同实现“高吞吐、低延迟”的核心目标。
3. 结合你具体参与的项目，说明高并发架构方案的选型依据、落地过程中的关键挑战及应对措施，以及最终的实施效果。

2024 年 6 月，我参加了“数据脱敏水印溯源统一管理平台”项目的建设，项目总投资 520 万元，历时 12 个月。该平台是为了解决企业在数据存储、数据共享等场景下面临的敏感数据泄露风险与合规压力问题，提升企业的数据安全合规与溯源能力。我在该项目中担任系统架构师，主要负责系统架构设计和技术选型等工作。

本文结合我的实践，以“数据脱敏水印溯源统一管理平台”项目为例，讨论高并发系统的架构设计与性能优化。接下来我将从弹性扩容、高并发查询优化、系统熔断降级三个维度进行论述。实践证明，通过以上技术的应用，能有效实现高并发系统的高吞吐、低延迟核心目标。最终系统于 2025 年 6 月正式上线，已接入近 100 个业务系统，日均处理脱敏请求 800 万次，获得业主单位年度优秀项目表彰。

随着《数据安全法》《个人信息保护法》等法规的全面实施，企业在数据共享、开发测试及外发等场景中，面临敏感数据泄露风险与合规压力。当前，各业务系统往往独立实现数据脱敏与水印嵌入功能，缺乏统一的技术标准和管控手段，导致脱敏算法不一致、处理结果不可控，水印信息格式各异、难以解析。更为关键的是，一旦发生数据泄露事件，由于缺乏标准化的水印溯源机制，无法追溯泄露源头、定责定损，给企业带来巨大的法律风险与经济损失。为解决上述问题，数据脱敏水印溯源统一管理平台应运而生。该系统有效实现了对敏感数据的统一识别、脱敏处理、水印注入及泄露溯源，全面提升企业数据安全防护能力与合规治理水平。2024 年 6 月，我参加了“数据脱敏水印溯源统一管理平台”项目的建设，项目总投资 520 万元，历时 12 个月。我在该项目中担任系统架构师，负责系统总体的架构设计、技术选型、核心模块技术攻关，带领架构团队共 5 人。系统的主要功能模块包括：涉敏数据治理、脱敏水印策略管理、算法密钥管理、溯源管理、运营监控等。该系统采用微服务架构，使用 Nacos、Redis、Kafka、PostgreSQL、ClickHouse 等技术，采用 DevOps 实现持续集成和交付，结合 Docker 和 Kubernetes 实现容器化部署和编排。该项目目标接入业务系统 500 个，因此存在瞬时流量洪峰，有高吞吐、低延迟的性能需求。

高并发系统的核心设计原则有弹性扩容 异步削峰 缓存分担流量压力 服务熔断降级



無諾

时间、项目介绍、岗位、职责、项目规模、历时都有，语言流畅



無諾

项目背景很完整，交代清楚了项目出现的原因



無諾

回应第一个子问题，对项目功能进行描述，以及对项目所涉及到的技术进行阐述



高并发系统的核心设计原则有弹性扩容、异步削峰、缓存分担流量压力、服务熔断降级等。在面对瞬时流量洪峰时，服务网关通过负载均衡将请求分发至不同的微服务，此时可以通过流量阈值监测，对微服务实例实现弹性扩容，以避免单节点被压垮。还可以引入消息队列，将请求发送至消息队列里，再根据流量情况动态增减分区数量、消费者数量，比如在我的项目里，脱敏日志通过 **Kafka** 异步写入 **ClickHouse**，以此实现了异步削峰。在微服务内部，为了避免数据库连接池资源耗尽和保护数据库不被瞬时流量冲垮，可以加入 **Redis** 作为缓存

来分担大部分的流量，只有极少部分流量才会进行到数据库。如果出现请求超时响应、资源耗尽等问题，可以通过熔断等机制实现服务的主动降级，返回默认值或页面友好提示稍后再试。通过以上技术手段，能在确保数据最终一致性的前提下，有效承接瞬时高并发流量，同时保障核心链路的稳定运行。以下结合“数据脱敏水印溯源统一管理平台”项目实际，详细论述我在实践中的具体做法。

关于弹性扩容的设计原则，我识别到脱敏请求量存在明显的峰谷，工作日白天高、夜间低，如果微服务采用固定副本数会造成资源浪费，高峰期又出现资源不足的情况。我提供了两种解决方案。一种是基于 **Kubernetes HPA** 实现的实时响应方案，根据 CPU、内存、网络带宽、磁盘 IO 等多个核心指标的阈值，实现自动扩缩容，白天高峰期脱敏服务最多有 20 个副本，夜间低谷时期缩至 3 个副本。多副本再配合 **Nacos** 集群的健康检查，自动摘除故障节点。另一种是基于定时的弹性伸缩方案，根据历史流量规律预设扩缩容时间表，8:00-17:00 保持 15 副本，其余时间保持 3 副本。对以上两种方案分析对比，方案一实时响应流量变化，适应突发流量，但存在扩容 1-2 分钟的滞后性，方案二无滞后、资源规划精准，但无法应对突发流量。最终我选择基于 **HPA** 的实时扩缩容方案，因为脱敏业务存在不可预测的突发高峰，实时响应能力比定时更关键。

关于高并发策略查询的性能优化，我引入 **Redis** 作为缓存提前加载热点策略。起因是业务高峰期，脱敏服务出现大量请求超时，服务不可用。经过排查分析，我发现策略查询接口的 QPS 峰值达到 2 万，PostgreSQL 数据库 CPU 100%，PostgreSQL 成为系统的性能瓶颈。于是



無諾

这是一个过渡句，但是略显生硬，应该想办法来通过描述这个项目来引出第二个问题



無諾

回答第二个子问题，在其中引入自己的项目来增加可信度以及让纯概念与你的项目进行结合，此处可以用一个真实例子来说明，更加增强说服力



無諾

论点的叙述，第一句最好是对论点的专业名词做解释，比如这里需要用 1-2 句话解释下什么叫弹性扩容，然后再写弹性扩容跟你项目相关的地方



無諾

后面的内容写的挺好，有业务、有背景、有技术选型、有技术实现

关于高并发策略查询的性能优化，我引入 Redis 作为缓存提前加载热点策略。起因是业务高峰期，脱敏服务出现大量请求超时，服务不可用。经过排查分析，我发现策略查询接口的 QPS 峰值达到 2 万，PostgreSQL 数据库 CPU100%，PostgreSQL 成为系统的性能瓶颈。于是我引入 Redis 作为缓存提前加载热点策略，查询逻辑为优先查缓存，如果未命中再查数据库，同时写回缓存。经过一段时间运行和指标监测，缓存命中率达到 95% 以上，数据库 CPU 降至 20%。但随着 Redis 的引入，随之而来的是数据一致性问题。最初我们采用的是“延迟双删”方案，但难以把握“等待的窗口时间”，等待时间太长影响接口性能，等待时间太短不能解决数据一致性。于是我决定部署 Canal 监听数据库 binlog，异步更新策略到 Redis 缓存。因为 Canal 保证消息的可靠和顺序，且时间延迟很低、对业务代码无入侵，所以不仅实现了数据的最终一致性，还提高了系统性能和可维护性。

系统有一个典型的高并发下的熔断降级场景。脱敏服务自身资源耗尽，如 CPU 满载、内存不足等，脱敏处理能力急剧下降，大量请求积压，可能导致系统雪崩。因此我决定引入熔断降级组件，在进行组件选型时，我对比了 Netflix 开源的 Hystrix 与阿里巴巴开源的 Sentinel。Hystrix 基于线程池隔离，线程切换开销较大，且近年进入维护状态，不再更新版本。Sentinel 以流量控制为切入点，采用轻量级信号量隔离，性能开销低于 Hystrix。Sentinel 还支持动态规则配置，可以通过 Nacos 实现实时推送规则，无需重启。而 Hystrix 的规则配置多为静态，调整后需要重新部署。综合性能、社区活跃度、动态规则配置考虑，最终我选择使用 Sentinel 组件配置熔断降级。具体做法是，针对脱敏服务设置系统负载阈值，CPU 使用率超过 90% 且持续时间 30 秒，触发熔断降级，脱敏服务快速返回预设的默认脱敏结果，对敏感字段统一返回***，保证核心链路的可用性。引入 Sentinel 后，脱敏服务在 CPU 满载时能够自动降级，用户几乎无感知。

整个项目历时 12 个月，于 2025 年 6 月完成交付，截至目前已稳定运行 10 个月，接入的业务系统已经近百个，累计处理脱敏请求 24 亿次，接入业务系统满意度达 96%，获业主单位信息安全管理年度优秀表彰。系统成功支撑了多次流量高峰，验证了高并发架构设计的有效性。经过该项目的实施验证，我深刻认识到，高并发场景下的架构设计需要贯穿需求分析、设计、开发、运维全生命周期。特别是系统可用性不是靠单节点高可靠实现的，而是靠冗余、自动故障转移、快速恢复共同实现。但在微服务治理方面还暴露了一些不足，比如服务拆分粒度难掌握，导致服务数量过多。后续计划引入领域驱动设计，通过划分限界上下文、区分子域类型、定义边界等原则，重新划分微服务，以提高系统的可维护性和可修改性。

后面的内容写的挺好，有业务、有技术选型、有技术实现



無諾

这句结尾的话是对整个文章的总结和收尾，是必须得有的



無諾

进行项目的收获的总结，以业务或者技术的方式进行叙述



無諾

引入一些客观存在的问题，提出对应的解决方案进行收尾



范文：论软件系统架构风格-反例



系统架构风格（**System Architecture Style**）是描述某一特定应用领域中系统组织方式的惯用模式。架构风格定义了一个词汇表和一组约束，词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。软件系统架构风格反映了领域中众多软件系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。软件系统架构风格的共有部分可以使得不同系统共享同一个实现代码，系统能够按照常用的、规范化的方式来组织，便于不同设计者很容易地理解系统架构。

请以“软件系统架构风格”论题，依次从以下三个方面进行论述：

1. 概要叙述你参与分析和开发的软件系统开发项目以及你所担任的主要工作。
2. 分析软件系统开发中常用的软件系统架构风格有哪些？详细阐述每种风格的具体含义。
3. 详细说明在你所参与的软件系统开发项目中，采用了哪种软件系统架构风格，具体实施效果如何。



2020 年 3 月本人参加了公司为某银行定制开发的集中作业处理系统。我在该项目中担任系统架构师，负责系统的架构设计工作。该系统是将银行各网点的柜面办理的录入功能转为一个或多个作业中心统一处理。该项目分为影像数据接收、影像切分、影像任务分发、任务处理和数据回传等功能。该项目历时 1 年时间于 2021 年 3 月正式上线，运行至今系统运行稳定，得到客户的认可和好评。

随着银行业务的发展，为提高效率，银行已全面完成改造，将传统的柜面业务进行简化，柜面只进行基本的业务操作（业务资料的确认），将繁重的、易出错的录入转由业务中心统一处理，大大的简化了柜台人员的工作，并且对数据监控、数据安全有了更好的保障。随着业务的进一步发展，银行又逐渐将此业务转包给具有此能力的公司执行。我公司多年专业从事银行业务，成功对接过多种银行业务系统。在业内享有较高的评价。并于 19 年底成功竞标成功的某银行的集中作业项目。

为了顺利完成此项目，公司特成立了此项目工作组，我在该项目组中担任系统架构师，负责系统的架构设计工作。作为银行系统，其特点是数据的保密性、系统的可用性和时效性都要求非常高。为了满足这些需求项目组确定了系统基本的技术选型，使用 netty 实现业务传输出框架，RocketMq 进行任务的分发和任务获取，使用 RAS 加密算法及进行数据传输过程的加解密处理，并使用 OCR 技术实现部分数据的自动识别。项目功能模块可分为影像数据接收、影像切分、影像任务分发、任务处理和数据回传等功能。从架构的角度思考，要将这些技术良好的组织在一起，首先就要选择好一组最合适的架构风格，在此基础上完成架构设计。



無諾

直接写某银行定制开发的就好，加上公司为这几个字有些多余。



無諾

应该写该系统是什么，然后它为银行提供了什么服务，语句有点不太流程哈。



無諾

你的摘要部分还是少了些，少了围绕中心论点的项目概述，这样的话，会有扣分的风险，因为摘要只介绍你和项目，没有介绍项目和论点的联系，可能你会觉得在子题目回应那儿有项目和论点的联系，但是摘要就是一个单独的部分。



無諾

注意检查错别字和病句。



常用的架构风格有，如下几个主要架风格。数据流体系结构风格、调用/返回体系结构风格、以数据为中心的体系结构风格、虚拟机体系结构风格、独立构建体系风格。数据流体系风格又包含有批处理体系风格和管道-过滤器风格，管道过滤器风格主要特点是数据是一个接一个的处理，前一个节点的输出作为下一个节点的输入，此风格常用在进行权限校验、数据校验的过滤器中。调用/返回体系结构风格主要思想是将复杂的大系统分解为若干的子系统，以降低复杂度，其又可分为主子程序风格、面向对象体系结构风格、层次行体系结构风格和客户端/服务器体系风格。以数据为中心的体系结构风格主要包括仓库体系风格和黑板体系风格。虚拟机体系结构风格可分为解释器体系风格和规则系统体系风格，规则体系风格可以预先定义号规则集通过输入的数据执行不同的规则。独立构建体系风格强调了系统中的每个构建都是相互独立的，它们之间不直接通信，主要包括进程通讯和事件系统风格。

结合银行系统的特点数据的保密性、系统的可用性和时效性都要求非常高，对改系统我们主要采用了层次体系结构风格、规则系统体系结构风格、客户端/服务器系统架构风格和进程通讯体系结构风格。

总体的架构我们采用的是层次体系结构风格。主要分为系统前置层、核心业务处理层、和应用业务服务层。核心业务处理层作为最底层模块主要负责主业务的数据处理，同时接收来自的前置系统的数据并向应用业务层提供基础数据接口。每一个层次都有自己独立的功能。服务内的功能修改不会对其他两个服务，真正作到了高内聚低耦合。

前置系统主要是负责与银行的数据进行交互，首先是接收来自银行系统的任务数据。为了保证数据的接收的快速处理，我们与银行之间确定了直接采用 socket 接口进行通讯以减少不必要的网络协议的校验。在这个技术选型上我们采用了已有的成熟框架 netty 作为底层代码。作为银行系统的数据是分有各种类型的，其中每个类型的数据业务处理要求也会有或多或少的区别。为了满足这个需求在前置系统中我们预先定制了一组规则，在接收到银行业务数据时根据数据的关键字与规则进行匹配，并作相应的规则处理。随着业务的发展只需对这组规则进行调整就可以完成新业务的匹配。将复杂的业务数据简化，做到代码不会因为业务的变更而不断的修改。前置系统作为银行系统的交互服务，数据传输的保密性非常重要，

因此前置系统采用了管道-过滤器的风格分别对传输数据的来源进行校验和传输数据的加解密。



無諾

如果字数不够，可以尝试在这一段为你要进行论述的架构风格解释那儿加上一个举例。



無諾

可以把每一层所使用的架构风格在介绍了这一层作用之后就说明使用了什么架构风格以及使用的原因你写在段落里面了，这样不好。



無諾

这一段把架构风格放这里来写就可以，但是建议先

密。

核心业务处理层主要采用了进程间通讯（隐式调用）的风格。核心业务处理层主要负责接收来自前置系统接收到的消息数据。接收到前置消息数据后核心业务处理层会将业务数据转换为任务数据，并将任务数据通过消息发送给业务处理前端进行处理。作为消息中间件的选型，本系统采用了 RocketMq，RocketMq 是阿里旗下的一款开源的项目，它承担阿里的双 11 的大数据量、高并发的业务处理能力。它的主要特色就是运行稳定，速度快且数据可保存在硬盘下，保证了数据的可靠性，它的这些特性非常符合银行系统的要求。它也成为系统上线后业务数据的高效运行起到了关键作用。

作为系统的最终业务处理模块，录入人员的前端处理模块我们采用了 C/S 的三层架构。作为前端采用了 C/S 架构而未采用目前主流的 B/S 架构的主要原因是，首先，此系统是专有的录入程序，使用者都是集中在一个作业中心进行处理，不存在用户分布在不同的区域。作业中心对软件的同一管理比较方便。第二 C/S 架构的客户端可以将基础数据保存在本地，减少了基础数据的获取时间，加快了业务处理能力。第三在客户端可方便的对一些敏感数据进行加密处理，可以更好的保证数据的保密性。第四此操作界面操作相对单一且功能固定，运行稳定很少会进行改动。客户端同应用业务服务层进行通讯，通过业务服务层获取核心业务层的录入任务消息。

该项目于 2021 年 3 月完成验收并正式投产使用，该项目运行至今，系统运行稳定，未出现大的生产问题，其中每天处理的业务量在 5 万左右，业务高峰期可达到每天 10 万左右的业务量，单位时间内处理的最大业务量为每分钟 300。业务处理得到了用户的认可。目前项目组留有 1 个维护人员进行系统日常维护工作。

系统的不足，作为银行系统，我们还是采用的传统的 ORACLE 数据库，虽然它的运行稳定，可处理的数据量相对较大。但相应的软硬件成本都偏高。目前对现有的技术对开源数据都有很好的解决方案，更换数据库可考率在下一步的系统升级改造中进行。

这一段把架构风格介绍完就可以，但是建议先介绍这一层的作用，然后就可以介绍这一层的特点以及采用的架构风格。

無諾
不要介绍技术产品，你可以把消息中间件进行简要介绍，如果字数不够，可以再就你们项目中所使用这个技术所碰到的问题以及怎么解决的来凑字数。

無諾
论文的这三段我们说，不仅仅要紧贴论点，要说项目，要说技术，还要说业务，你这三段业务说的太少，真实性就会大大的打折扣，而且我说过，为了提高可信度，可以加上一些可量化的指标、数据进行支撑。

無諾
这些数据可以通过更加专业点的写法或者英语单词来描述，凸显专业度。

無諾
结尾可以加上一些经验之谈或者是不足之处和解决的方式，不太建议写这种系统还有一些不好的地方，我们后面再升级这种话。

T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

