

芝士系分案例冲刺红宝书

2025 年 11 月 - 芝士架构凯恩编辑整理 v3.0.0



目录

芝士系分案例冲刺红宝书	1
前言	10
考试规则	10
命题趋势	11
刷题形式	12
案例题型	15
关注范围	16
1. 上篇-需求分析（次重点★★★☆☆）	17
1.1. 结构化分析（重点★★★★★）	17
1.1.1. 数据流图 DFD（重点★★★★★）	17
1.1.2. 状态转换图（次重点★★★☆☆）	21
1.2. 面向对象分析方法（超级重点★★★★★）	24
1.2.1. UML 引入（重点★★★★★）	24
1.2.2. 用例图（超级重点★★★★★）	25
1.2.3. 类图（重点★★★★★）	32
1.2.4. 顺序图（重点★★★★★）	36
1.2.5. 通信图/协作图（重点★★★★★）	39
1.2.6. 活动图（重点★★★★★）	40
1.2.7. 常见图对比（重点★★★★★）	47
2. 上篇-系统设计/面向对象设计（重点★★★★★）	47
2.1. 类识别（次重点★★★☆☆）	48

2.1.1. 识别原则（次重点★★★☆☆）	48
2.1.2. 典型例题（重点★★★★★）	48
2.2. 面向对象设计原则（重点★★★★★）	51
2.2.1. 设计原则（重点★★★★★）	52
2.2.2. 典型例题（重点★★★★★）	52
3. 上篇-数据库系统基础（超级重点★★★★★）	53
3.1. 关系模型（重点★★★★★）	53
3.1.1. 关系运算基础（次重点★★★☆☆）	53
3.2. 数据库设计与建模（超级重点★★★★★）	57
3.3. 概要设计（超级重点★★★★★）	58
3.3.1. 整体 E-R 图设计（重点★★★★★）	60
3.4. 逻辑设计（超级重点★★★★★）	60
3.4.1. E-R 图转换关系模式（重点★★★★★）	61
3.4.2. 函数依赖（重点★★★★★）	62
3.4.3. 无损联接分解（次重点★★★☆☆）	63
3.4.4. 保持函数依赖（次重点★★★☆☆）	64
3.4.5. 规范化意义（重点★★★★★）	65
3.4.6. 键/码/属性（重点★★★★★）	65
3.4.7. 五大范式（超级重点★★★★★）	67
3.4.8. 典型例题（超级重点★★★★★）	71
3.5. 数据库的控制功能（重点★★★★★）	74
3.5.1. 并发控制（重点★★★★★）	74

3.5.2. 事务的基本概念（重点★★★★★）	74
3.5.3. 数据一致性问题（重点★★★★★）	75
3.5.4. 封锁技术（重点★★★★★）	76
3.6. 数据库性能优化（次重点★★★☆☆）	78
3.6.1. 反规范化（重点★★★★★）	78
3.6.2. 索引优化（重点★★★★★）	79
3.6.3. 完整性约束（重点★★★★★）	79
3.6.4. 触发器示例（重点★★★★★）	80
3.6.5. 典型例题（重点★★★★★）	80
3.7. 备份与恢复技术（次重点★★★☆☆）	83
4. 上篇-Web 与移动应用系统分析与设计（新教材★★★★★★）	84
4.1. Web 设计原则（超级重点★★★★★）	84
4.2. Web 架构技术选型（超级重点★★★★★）	85
4.3. 其他架构模式（次重点★★★☆☆）	86
4.3.1. 和视图相关的架构（次重点★★★☆☆）	86
4.3.2. P2P 架构（次重点★★★☆☆）	90
4.4. Web 应用部署（重点★★★★★）	91
4.4.1. Web 应用部署原则（重点★★★★★）	91
4.5. 典型例题（超级重点★★★★★）	91
5. 上篇-大数据架构（新教材★★★★★）	93
5.1. 系统架构原则（次重点★★★☆☆）	93
5.2. 批处理架构 VS 流处理架构（重点★★★★★）	94

5.3. Lambda 架构 (重点★★★★★)	95
5.3.1. Lambda 分层介绍 (重点★★★★★)	95
5.3.2. Lambda 架构实现 (重点★★★★★)	97
5.3.3. Lambda 架构优缺点 (重点★★★★★)	98
5.4. Kappa 架构 (重点★★★★★)	98
5.4.1. Kappa 架构的优缺点 (重点★★★★★)	99
5.4.2. Lambda VS Kappa (超级重点★★★★★)	100
5.5. IOTA 架构 (次重点★★★☆☆)	100
5.6. 大数据系统开发 (重点★★★★★)	102
5.6.1. 数据存储 (重点★★★★★)	102
5.6.2. 数据管理 (次重点★★★☆☆)	103
5.6.3. 数据处理 (重点★★★★★)	104
5.7. 典型架构 (重点★★★★★)	105
5.7.1. 某网奥运 (重点★★★★★)	105
5.7.2. 某网络广告平台 (重点★★★★★)	107
5.7.3. 某证券公司日志大数据系统 (重点★★★★★)	108
5.7.4. 某电商智能决策大数据系统 (重点★★★★★)	110
5.8. 典型例题 (重点★★★★★)	111
6. 上篇-云原生架构设计 (超级重点★★★★★)	114
6.1. 云原生架构原则 (重点★★★★★)	114
6.2. 云原生架构模式 (重点★★★★★)	114
6.3. 不好的实践 (重点★★★★★)	115

6.4. 容器技术（重点★★★★★）	116
6.4.1. 容器和虚拟机对比（重点★★★★★）	116
6.4.2. 容器和容器编排技术（重点★★★★★）	117
6.4.3. 容器运维指令（重点★★★★★）	118
6.4.4. 典型例题（重点★★★★★）	119
7. 上篇-信息安全（超级重点★★★★★）	120
7.1. 数据加密技术（超级重点★★★★★）	120
7.1.1. 对称加密概念（超级重点★★★★★）	121
7.1.2. 非对称加密概念（超级重点★★★★★）	121
7.1.3. 对称加密和非对称加密的区别（超级重点★★★★★）	122
7.1.4. 典型例题（超级重点★★★★★）	122
7.1.5. 数字签名（重点★★★★★）	123
7.1.6. 数字证书（重点★★★★★）	125
8. 上篇-微服务模式（新教材★★★★★）	126
8.1. 微服务设计原则（论文用得到）	126
8.2. 微服务设计模式（重点★★★★★）	127
8.3. 微服务开发（重点★★★★★）	129
8.3.1. 服务注册和发现简介（次重点★★★☆☆）	130
8.3.2. 服务通信（超级重点★★★★★）	130
8.3.3. SpringCloud 微服务框架图（重点★★★★★）	132
8.4. 安全与权限管理（重点★★★★★）	134
8.4.1. OAuth 2.0（重点★★★★★）	134

8.4.2. JWT (重点★★★★★)	135
8.4.3. API 网关 (非重点☆☆☆☆☆)	136
9. 上篇-面向服务架构 (超级重点★★★★★)	136
9.1. 什么是 SOA (重点★★★★★)	136
9.2. SOA/微服务/单体架构对比 (超级重点★★★★★)	137
9.2.1. 微服务和单体架构对比 (超级重点★★★★★)	137
9.2.2. 微服务和 SOA 对比 (超级重点★★★★★)	138
9.2.3. SOA 和单体架构对比 (超级重点★★★★★)	139
9.3. 服务注册模式 (重点★★★★★)	140
9.3.1. 服务注册模式概念 (重点★★★★★)	140
9.3.2. UDDI、WSDL 和 SOAP (重点★★★★★)	140
9.3.3. REST 规范/风格 (重点★★★★★)	141
9.3.4. 典型例题 (重点★★★★★)	142
9.4. ESB 模式 (重点★★★★★)	144
9.4.1. ESB 概念 (重点★★★★★)	144
9.4.2. 典型例题 (重点★★★★★)	145
9.5. SOA 设计原则 (重点★★★★★)	145
10. 上篇-系统规划 (重点★★★★★)	146
10.1. 项目立项方法 (次重点★★★☆☆)	146
10.2. 可行性研究 (重点★★★★★)	146
10.2.1. 可行性研究的四个方面 (重点★★★★★)	147
10.2.2. 典型例题 (重点★★★★★)	147

10.3. 成本和收益（次重点★★★☆☆）	148
10.3.1. 概念解释（次重点★★★☆☆）	148
10.3.2. 典型例题（重点★★★★★）	148
10.4. 成本效益计算（重点★★★★★）	149
10.4.1. 公式汇总（重点★★★★★）	149
10.4.2. 净现值（重点★★★★★）	150
10.4.3. 净现值率（重点★★★★★）	150
10.4.4. 投资回收期与投资回报率（重点★★★★★）	151
10.4.5. 典型例题（重点★★★★★）	153
11. 上篇-系统架构设计（次重点★★★☆☆）	155
11.1. 软件架构风格（重点★★★★★）	155
11.1.1. 概念辨析（重点★★★★★）	155
11.1.2. 典型例题（重点★★★★★）	156
11.2. 面向架构评估的质量属性（次重点★★★☆☆）	157
12. 下篇-数据库系统（重点★★★★★）	158
12.1. 数据库性能优化（重点★★★★★）	158
12.1.1. 索引建立注意事项（重点★★★★★）	158
12.1.2. 数据分片（重点★★★★★）	159
12.1.3. 数据分区（重点★★★★★）	161
12.1.4. 分库分表（重点★★★★★）	162
12.2. 数据库高可用模式（重点★★★★★）	163
12.2.1. 读写分离（重点★★★★★）	163

12.2.2. 主从复制（重点★★★★★）	164
12.2.3. 一致性问题（重点★★★★★）	166
12.3. 数据库的选型（重点★★★★★）	167
13. 下篇-分布式系统（重点★★★★★）	168
13.1. CAP 理论（重点★★★★★）	168
13.1.1. 数据一致性（重点★★★★★）	169
13.1.2. 可用性（重点★★★★★）	170
13.1.3. 分区容错性（重点★★★★★）	171
13.2. CAP 理论与系统实现（重点★★★★★）	173
13.3. BASE 理论（重点★★★★★）	173
13.4. 分布式事务理论解决方案	174
13.4.1. 刚性事务 2PC（重点★★★★★）	175
13.4.2. 柔性事务 TCC（重点★★★★★）	177
13.4.3. 柔性事务本地消息表（重点★★★★★）	179
14. 后记	181

前言

各位同学好，众所周知，案例分析是架构三个科目中最难的最容易翻车的科目。之所以难，一是因为案例出题范围不固定，导致我们准备起来很难，很难精准打击。二是因为需要记记背背的东西不少，假如你只是看，不去记忆，考试的时候遇到题目写不出一二三来还是会翻车。所以，针对这两个问题，凯恩根据自己对软考高级的理解，结合官方教材、历年真题和权威教材编制了这本案例冲刺红宝书。这本红宝书的目标就是帮你缩小准备范围，帮你节约 30% 的时间，让你在做题目的时候能够有话可说。

版权提醒：本资料已通过国家版权局登记（登记号：渝作登字-2025-A-00624260），（一旦发现资料恶意流出，直接封号不退款，请珍惜账号权益）。

考试规则

从 2024 年开始，案例考试和综合知识被放在上午一起考。这两门考试加起来是 240 分钟，其中综合知识最短作答时间是 120 分钟，最长 150 分钟。按照以往经验，综合题时间比较充裕，大部分同学都可以结余出 30 分钟左右的时间。按照新规你可以提前交卷，多出来的这 30 分钟可以和案例原本的 90 分钟累加，最多有 120 分钟的时间来做案例。

因为案例相对选择时间紧题目难，90 分钟搞定会比较吃力，所以凯恩在这里诚恳地建议各位同学选择题至少提前 15 分钟交卷，结余 15 分钟放到案例分析中。

综合和案例分析联考之后时间安排（合计 240 分钟）			
综合	最短 120 最长 150 分钟（120 分钟后可以交卷）	满分 75 分	75 题
案例分析	最短 90 分钟最长 120 分钟（选择节约的时间可以放到案例分析）	满分 75 分	5 题第一题必做题，后面 4 选 2

从表格里我们可以看到，系统架构设计师和系统分析师的案例分析和高项完全不同，它是有选做题的。考试的时候，机考作答系统会给你 5 道题目。其中第一题是必做题，后面四题四选二。这样必做一题加上选做两题等于三道题，每题的分值是 25 分，就可以凑成 75 分的总分。

之所以有选做题，是因为架构系分设计之初是要照顾不同行业背景的软件从业者，所以考试主题涵盖数据库、大数据、嵌入式等等，让考生根据自己的行业背景来做题。这个出发点当然是好的，但是目前来看，实际上考察的内容还是偏向 JavaWeb 居多。

命题趋势

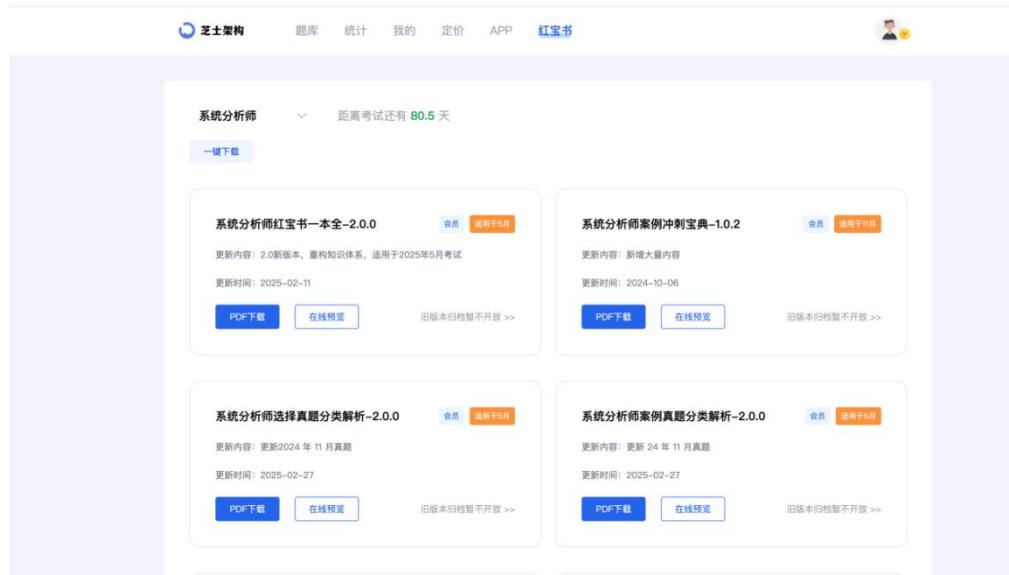
案例虽然离谱，但还是有一定的规律。实际上结合近两年来的考试情况可以发现，无论架构还是系分，案例第一题考察的知识点必定是书本上的内容。第一题 25 分只要你用心背，至少可以拿 20 分。而其余 4 道题目，确实极难预测，但基本分布在需求分析、数据库、缓存等主题范围内。对于这些主题，红宝书会尽可能深入且全面地呈现相关内容，力求押中题目，让你可以拿到剩余的 25 分，45+通过考试。

所以结合上面的知识点分布形式，凯恩将案例红宝书分为上篇和下篇，上篇

主要提炼书本上的案例相关考点，下篇主要聚焦书本之外的内容，力求提升你的技术视野，让您做选做题的时候有话可说。

刷题形式

案例一定要做真题，否则你不知道题目形式、答题方法和自己的真实水平。凯恩建议每个同学都要去练真题做真题至少三遍。做真题有两个渠道，第一个去官网下载案例分类解析，这里的题目凯恩按照知识点大类对内容做了全面的解析，方便你对知识点进行重点突破。



第二是点击我们的官网首页 www.cheiko.cc 的历年真题模块，下载历年试题打印出来进行练习（都是附带解析的）。

当前位置 – 历年真题

系统分析师 考试时间预计为 2025-05-24，距今还有 80.5 天，报名开始时间预计为 2025-03-25

备考红宝书 14万字 – 浓缩备考知识点
20万字 – 详细历年真题解析

学习互动群 无限次-备考详细答疑互动交流
累计3次-论文批改及写作指导

选择题 案例题 **论文题**

一键下载

2024年11月 (案例题真题) 导出 ↗ 11月 难度简单 掌握程度 ★★★★★ 最近得分 0分 / 满分 11分 去练习 →

2024年5月 (案例题真题) 导出 ↗ 5月 难度中等 掌握程度 ★★★★★ 最近得分 0分 / 满分 11分 去练习 →

2023年5月 (案例题真题) 导出 ↗ 5月 难度简单 掌握程度 ★★★★★ 最近得分 0分 / 满分 13分 去练习 →

当然假如你想在线练习也可以，点击官网案例解析模块在线练习也非常容易（缺点是不好做笔记）。

系统分析师 考试时间预计为 2025-05-24，距今还有 80.5 天，报名开始时间预计为 2025-03-25

错题记录 收藏记录 历年真题 练习日志

备考红宝书 14万字 – 浓缩备考知识点
20万字 – 详细历年真题解析

学习互动群 无限次-备考详细答疑互动交流
累计3次-论文批改及写作指导

选择题 **案例题** 论文题

请输入需要查询的题目 搜索

软件工程 不看必挂 暂无要点 去练习 →
掌握程度 ★★★★★ 刷题进度 0已做/16合计

Web开发 不看必挂 暂无要点 去练习 →
掌握程度 ★★★★★ 刷题进度 0已做/37合计

数据库系统 不看必挂 暂无要点 去练习 →
掌握程度 ★★★★★ 刷题进度 0已做/34合计

系统分析与设计 不看必挂 暂无要点 去练习 →
掌握程度 ★★★★★ 刷题进度 0已做/16合计

系统规划 非重点 暂无要点 去练习 →
掌握程度 ★★★★★ 刷题进度 0已做/16合计

嵌入式系统设计 非重点 暂无要点 去练习 →
掌握程度 ★★★★★ 刷题进度 0已做/16合计

The screenshot shows a web-based learning platform for case studies. At the top, there is a navigation bar with links for '芝士架构' (Cheesecake Architecture), '题库' (Question Bank), '统计' (Statistics), '我的' (My Profile), '定价' (Pricing), 'APP', '红宝书' (Red Book), and a user profile icon.

The main content area displays a reading passage titled '2012年5月第2题'. The passage discusses a large consulting company's plan to upgrade its core business system to better serve individual users. It mentions two employees, Wang and Li, who proposed different solutions. Wang suggested using object-oriented technology and Java, while Li proposed a service-oriented architecture. After discussion, Li's solution was adopted due to its cost-effectiveness. The reading ends with a note about the complexity of the original system.

Below the reading is a question card for '问题 1' (Question 1). The question asks for a 500-word analysis comparing the two proposals and explaining why Li's was chosen. There is also a note indicating that the question is worth 10 points.

On the right side of the page, there is a sidebar with a timer set at 00:02, a font size selector (A 字号), and a date range selector (日间). Below these are several small boxes representing other questions from previous years, such as '12年5月第2题第1问' through '18年5月第5题第3问'. At the bottom of the sidebar, there is a navigation bar with arrows and the number '1 / 3'.

案例题型

从形式角度上来看，虽然官方没有给出题型类别，但是根据真题的出题方式，目前可以分为选词填空型、简答填空型、概念对比型、概念简述型、解决方案简述型五大类别，如下表所示。

案例题型	真题问法	难易程度
选词填空型	胰岛素剂量传输不准会带来一些安全问题，请将下列内容填入下方的空格处，完善可能导致胰岛素剂量不准的原因。 备选选项：(a) 血糖传感器错误 (b) 传输系统异常 (c) 血糖计算不准 (d) 定时器失效 (e) 泵信号失效 (f) 错误时间推送预定的量 (g) 算法错误 (h) 计算错误 (i) 胰岛素计算错误	简单
简答填空型	根据题干中描述的基本功能需求，架构师王工通过对需求的分析和总结给出了无人直升机控制系统纵向控制状态图。请根据题干描述，提炼出相应状态及条件，并完善如图所示状态图中的(1)~(5)，将答案填写在答题纸中。	较难
概念对比型	该系统需实现用户终端与服务端的双向可靠通信，请用 300 字以内的文字从数据传输可靠性的角度对比分析 TCP 和 UDP 通信协议的不同，并说明该系统应采用哪种通信协议。	较难
概念简述型	请用 300 字以内的文字，说明 Redis 分布式存储的两种常见方案，并解释说明 Redis 集群切片的几种常见方式。	最难
解决方案简述型	经过运维人员的深入分析，发现存在两种情况： (1) 用户请求的 key 值在系统中不存在时，会查询数据库系统，加大了数据库服务器的压力；(2) 系统运行期间，发生了黑客攻击，以大量系统不存在的随机 key 发起了查询请求，从而导致了数据库服务器的宕机。 经过研究，研发团队决定，当在数据库中也未查找到该 key 时，在缓存系统中为 key 设置空值，防止对数据库服务器发起重复查询。请用 100 字以内文字说明该设置空值方案存在的问题，并给出解决思路	最难

选词填空型是最简单的，因为给出的信息很多很直接，比较利于你做判断，对行业背景知识要求不高，这种题目是非后端类同学的最爱。

简答填空题是较难的，因为实际上这类题目也给你提供了一些信息，但是由于是开放的填空，所以比选择填空信息要少，难度略有提升。

概念对比型是较难的，一来因为是简述题，需要你结合自己的理解来写，同时要对比两个或者多个不同的工具和技术的异同，对你的知识储备要求较高。但是因为一般来说题目中可能会给你一些提示，比如从哪些方面进行比较，因此在作答的时候范围还是受到了约束，可以通过题干的倾向推断出一部分来。

解决方案简述型和概念简述型是最难的，因为几乎不给你额外信息，就是考你知识储备，假如没有接触过，或者没有准备过，一点都写不出来。

在考试的时候，你肯定要大致估算出每个大题的预测得分情况，选择一个总分最高的问题回答。

关注范围

因为今年是系分新教材的第一年，所以我们必须要重点关注新教材案例篇出现的概念，第一年必考必出。这些内容凯恩都标记了超级重点。嵌入式和信息物理系统一般来说不会放在第一题或者重点考察，感兴趣的看一下教材了解即可。新增的移动应用开发，凯恩看了好多遍，实在想不出它能怎么考，主要介绍了框架、语言，涵盖 APP、小程序等技术，很杂不深，所以也不体现。实际上系分的考点还是相对可控，凯恩有信心大部分的内容都在红宝书提及，少部分真题出现过的概念比如区块链，消息队列等通过真题补充。

另外，系分架构的案例永远有一个永恒的主题，就是技术选型和对比。一个技术一个框架它绝不是十全十美的，它有哪些优势哪些劣势，适用范围在哪里，在接触一个技术新词的时候，要有挖掘这些信息的能力。本红宝书的典型例题包含了架构还有系分，这个都要看。

1.上篇-需求分析（次重点★★★☆☆）

这一章节的考察主要还是会以需求建模（填空）为主，加上对数据流图，状态转换图，数据字典，UML 图等相关概念的考察。建模题你做过真题就知道，需要一定的逻辑，通过文字说明给出业务流程，让你在对应的状态图，顺序图，活动图上填空。这种题目往往都有争议，毕竟题干描述的需求往往比较模糊，需要你结合自己的理解去推理补全拼图，拿分容易，高分难。系分这块特别爱考，但是难度不大，只要出这个章节的题目就是送福利，送你过。

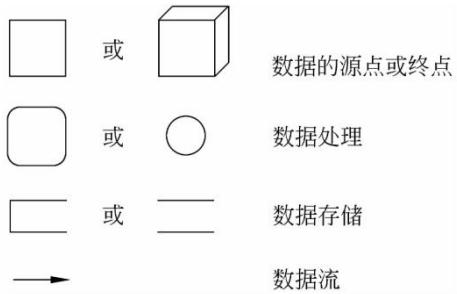
1.1.结构化分析（重点★★★★★）

1.1.1.数据流图 DFD（重点★★★★★）

1.1.1.1.基本符号（重点★★★★★）

在 DFD 中，通常会出现 4 种基本符号，如下表所示（结合图片进行记忆）。这一块是看图的基础，必须掌握。在选择题一本全里，我们重点讲到过这个部分，但是一般来说案例不会直接考你这些数据元素的内容，而是结合实际案例考察你对数据流图的运用。

符号名称	表示内容	图形表示
数据流	具有名字和流向的数据	标有名字的箭头
加工（数据处理）	对数据流的变换	圆圈
数据存储	可访问的存储信息	直线段
外部实体(数据源及数据终点)	位于被建模的系统之外的信息生产者或消费者，表明数据处理过程的数据来源及数据去向	标有名字的方框



补充：数据存储要被外部实体使用必须要经过加工。有一年案例考察过这个知识点。

有的同学看到其他资料反馈图对不上，这里特此解释。数据流图模板有两种常见的记号集：Yourdon 和 Coad 方法以及 Gane 和 Sarson 方法。这两个系统都以创造它们的计算机科学家的名字命名。上图左侧的是 Gane 和 Sarson 方法，右侧是 Yourdon 和 Coad 方法。

1.1.1.2. 平衡原则（重点★★★★★）

重点关注，数据流图的平衡原则，如下所示，要掌握父子图的平衡和子图的平衡，结合案例真题能够写出错误的理由或能写出基本的数据流图平衡办法。

（1）父图（上层数据流图）与子图（下层数据流图）平衡（也叫作分层细化过程平衡）。
个数一致：两层数据流图中的数据流个数一致。方向一致：两层数据流图中的数据流方向一致。

（2）子图内部的平衡表示如下。

概念	描述
黑洞	加工只有输入没有输出（只进不出）
奇迹	加工只有输出没有输入（只出不进）
灰洞	加工中输入不足以产生输出（加工应该输出 A，实际给出 B，挂羊头卖狗肉的意思）
数据存储	一般来说这个点可以不提。正常情况下必须既有读的数据流，又有写的数据流；在某张子图中，可能只有读没有写，或者只有写没有读。

1.1.1.3. 数据流图纠错（重点★★★★★）

数据流纠错是常考的难点和重点，除了上面的平衡原则，还要考虑到前面提到的基本符号是否运用准确，还要适当填写编号辅助理解，有的时候还要结合题干判断是否有数据流缺失等问题。

1.1.1.4. 典型例题（重点★★★★★）

（2023 年系分真题）阅读以下关于企业信息系统结构化分析的叙述，回答问题 1 和 2。

某软件公司为企业开发一套员工在线教育系统支持员工利用业余时间开展专业技术培训，提升员工技能。在项目开展初期，采用结构化分析进行开发，并对系统中培训部员工和培训教师的相关功能进行分析，具体需求如下：

（1）培训部根据企业技术发展需求，负责策划培训课程，并形成课程计划，针对不同的员工设置不同的课程；

（2）员工首先在系统进行注册，填写自己的编号，学历，专业，岗位等信息，生成员工注册信息，然后将自己的培训需求录入系统，系统自动评估并进行课程推荐，员工确认后形成课程需求；

（3）培训教师也通过系统进行注册，填写自己的编号、学历、专业等信息，形成培训教师注册信息

（4）系统根据课程计划、员工注册信息，课程需求和培训教师注册信息，为员工和培训教师生成对应的课程表。工时系统分析师对上述流程进行了审核，并指出需补充数据字典，从而更完整地对系统建模。

问：项目组针对题干描述的业务需求，初步绘制了系统流图（2-1），请分析图中的至少三类错误并对每类错误进行简单解释。

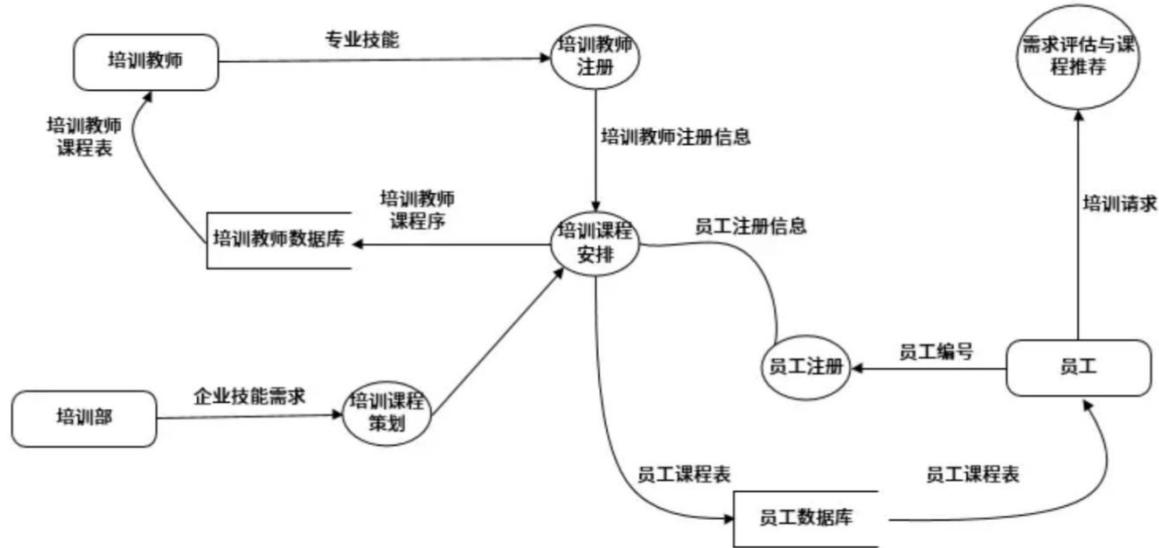


图 2-1 员工在线教育系统数据流图

这一题考察 DFD 图找茬。

我们先看数据流图，在绘制和审查 DFD 时，需要确保图中的每个元素都符合结构化分析的标准。DFD 的核心目标是清晰地表达系统的数据流动和处理过程，因此在设计 DFD 时必须遵循一定的规则：

1. 每个加工都必须有输入和输出（平衡问题）。
2. 数据流要有明确的方向，数据的流动路径必须清晰（图例问题）。
3. 每个加工和数据流都应有清晰的命名，以确保图形的可读性（命名问题）。
4. 编号系统帮助在不同的层次之间建立清晰的关系，避免混淆（编号问题）。

第 1 类错误： 加工只有输入没有输出。“需求评估与课程推荐”这个加工只有输入数据流，但是没有任何输出数据流。在 DFD 中，每个加工过程必须至少有一个输入流和一个输出流。如果一个加工没有输出流，就说明这个过程没有任何可见的效果，违反了数据流图的基本要求。

第 2 类错误： 数据流没有方向箭头。“员工注册信息”这一数据流没有标明方向箭头。数据流必须明确指示数据从哪里流向哪里，没有方向箭头会使得数据流的起点和终点不明确，从而

影响图形的理解和准确性。数据流必须具备方向性，指明数据传递的路径。

第 3 类错误：所有的加工没有编号。"需求评估与课程推荐"和"培训教师注册"等加工没有编号。在 DFD 中，每个加工都应有一个编号，编号可以帮助分析人员和开发人员跟踪和引用这些过程，特别是当 DFD 分层时，编号能够帮助关联不同层次之间的流程和数据。

第 4 类错误：数据流没有名称。"培训课程规划"到"培训课程安排"的这条数据流没有命名。在 DFD 中，每条数据流都必须有明确的名称，以便准确表示数据的含义和作用。没有名称的数据流使得图形不够清晰，影响其可读性和准确性。

第 5 类错误：不能从一个存储直接到另一个存储。"员工课程库"到"员工实体"之间的直接数据流是不允许的，同样 "培训教师数据库" 到 "培训教师实体" 也不能直接连接。数据存储和数据实体之间的流动应该通过加工来进行转换，而不能直接连接。因为数据存储是静态的，它们本身不进行处理，数据流动应通过相应的处理过程进行管理。

1.1.2. 状态转换图（次重点★★★☆☆）

状态转换图/状态机图，就是 UML1.x 中的状态图，利用状态和事件描述对象本身的行为。在书本上它是被归类到结构化分析工具里头的，实际上面向对象分析也用得到。它是一种非常重要的行为图，强事件导致的对象状态的变化。状态机图中的主要概念包括以下 3 个。（1）状态、初态、终态。（2）事件、转移、动作。（3）并发状态机。状态机图的推荐使用场合：包括类设计场合。目前只在系分中考过一次，架构案例还没考到过，可以准备一下。和数据流图一样，只会考你填空。

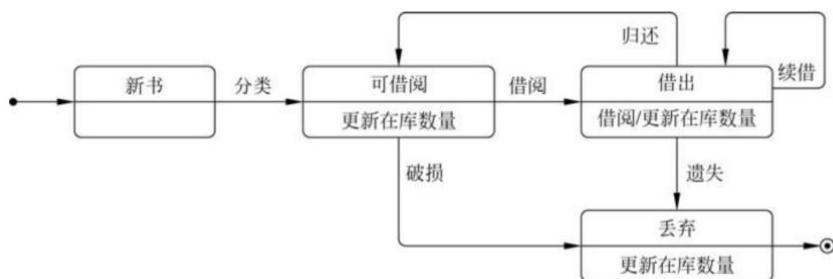
1.1.2.1. 基本符号（重点★★★★★）

状态转换图是一种描述系统对内部或外部事件响应的行为模型。它描述系统状态、事件和

事件引发系统在状态之间的转换。在状态转换图中定义的状态主要有：初态（即初始状态）、终态（即最终状态）和中间状态。初态用黑圆点表示，终态用黑圆点外加一个圆表示，状态图中的状态用圆角四边形表示，状态之间状态转换用带箭头的线表示。如下表所示。

元 素	图形符号	元 素	图形符号
状态(State)	State	初态(Initial State)	●
浅度历史(Shallow History)	(H)	终态(Final State)	◎
深度历史(Deep History)	(H*)	转移(Transition)	$e[g]/a \rightarrow$
选择(Choice)	◇	分叉/合并(Fork/Join)	—

例子如下：



1.1.2.2. 典型例题（重点★★★★★）

(2020 年系分真题) 某公司长期从事嵌入式系统研制任务，面对机器人市场的蓬勃发展，公司领导决定自主研制一款通用的工业机器人。王工承担了此工作，他在广泛调研的基础上提出：公司要成功地完成工业机器人项目的研制，应采用实时结构化分析和设计 (RTSAD) 方法，该方法已被广泛应用于机器人顶层分析和设计中。

下图给出了机器人控制器的状态转换图，其中 T1-T6 表示了状态转换过程中的触发事件，请将 T1-T6 填到图中的空 (1) ~ (6) 处，完善机器人控制器的状态转换图，并将正确答案填写在答题纸上。

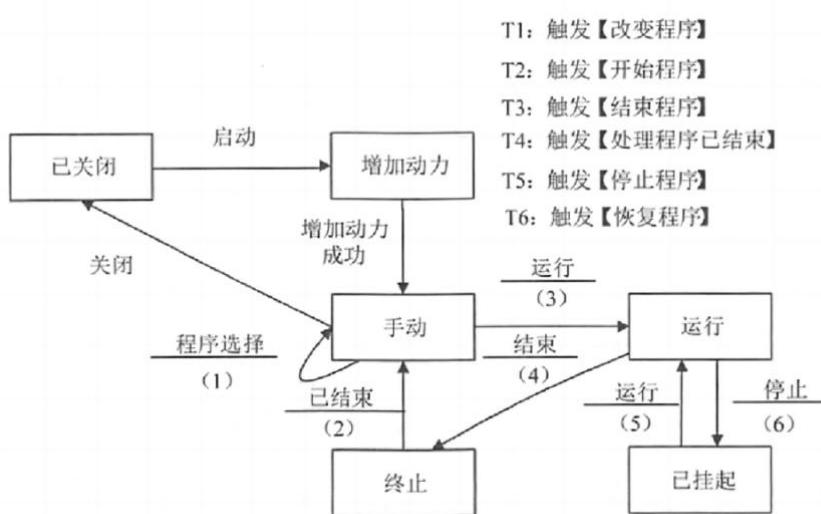


图 3-1 机器人控制器状态转换图

状态转换图，即 STD 图（StateTransformDiagram），表示行为模型。STD 通过描述系统 的状态和引起系统状态转换的事件，来表示系统的行为，指出作为特定事件的结果将执行哪些动作（例如处理数据等）。STD 描述系统对外部事件如何响应，如何动作。在状态转换图中，每一个节点代表一个状态。有题干可知，机器人控制器设定了 6 种状态，即已关闭、增加动力、手动、运行、终止和已挂起，在 6 个状态相互转换时，设计了 6 个触发事件（T1~T6）。

当按下启动按键时，系统就会进入增加动力状态。在成功地完成了增加动力的过程之后，系统就会进入手动状态。系统手动状态时操作员按下运行按钮，就会启动当前选择程序的执行过程，然后系统就会过渡到运行状态，所以第三空应该为 T2: 触发【开始程序】。

系统运行状态时操作员可以通过按下停止按钮来挂起程序的执行过程，然后系统就会进入已挂起状态，所以第六空应该为 T5: 触发【停止程序】。

系统已挂起状态时操作员可以按下运行按钮来继续执行程序，系统则返回到运行状态，所以第五空应该为 T6: 触发【恢复程序】。

系统运行状态时操作员可以按下结束按钮，系统进入终止状态，所以第四空应该为 T3: 触发【结束程序】。

当程序终止执行时要想返回手动状态，就需要触发【处理程序已结束】，从而回到手动状

态。所以第二空应该为 T4：触发【处理程序已结束】。

系统手动状态时操作员现在可以使用程序选择旋钮开关来选择程序，所以应该触发【改变程序】，第一空应该为 T1：触发【改变程序】。

答案：(1) T1 (2) T4 (3) T2 (4) T3 (5) T6 (6) T5

1.2. 面向对象分析方法（超级重点★★★★★）

面向对象分析方法架构系分都爱考常考，概念对比类、简答类、选词填空类都考过。这一小节，凯恩把常考的概念对比贴出来，并且结合典型例题，让你掌握这里的重点难点。

1.2.1. UML 引入（重点★★★★★）

软考里面向对象分析工具就是 UML。只要出题人家庭还和睦，情绪还稳定，那只会考类图、用例图、顺序图、协作图、活动图和状态图（之前结构化分析提到过）。别看内容少，但是这些图只要配合上任意场景就能摇身变成一道新的题目。另外这些图之间的对比异同也是它常考的内容，需要重点关注。下图是 UML2.0 包含的所有分析建模工具，但是常考的就是灰色高亮标注的这些，其他工具不需要深入，知道它们大致能做什么就行。

有人是大纲信徒，说大纲没有 UML，我不准备我不背了。软考的大纲和厕纸没有区别，现在还有这样的人你们就把它自动归类为傻子，让他们老老实实做分母吧。

名称	描述
类图	描述类、接口、协作及它们之间的关系
对象图	描述对象及对象之间的关系
包图	描述包及包之间的相互依赖关系

组合结构图	描述系统某一部分（组合结构）的内部结构
构件图	描述构件及其相互依赖关系
部署图	展示构件在各节点上的部署
外廓图	展示构造型、元类等扩展机制的结构
顺序图	示对象之间消息的交互，强调消息执行顺序的交互图
通信图/协作图	展示对象之间消息的交互，强调对象协作的交互图。协作图是 UML1 的说法。
时间图/定时图	展示对象之间消息的交互，强调真实时间信息的交互图
交互概览图	展示交互图之间的执行顺序
活动图	描述事物执行的控制流或数据流
状态机图	描述对象所经历的状态转移
用例图	描述一组用例、参与者及它们之间的相互关系

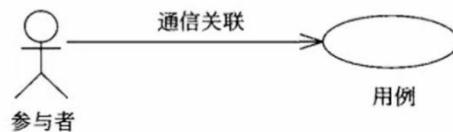
1.2.2. 用例图（超级重点★★★★★）

用例模型中只会结合具体场景考你识别参与者、调整用例模型的基本能力。这里要知道面向对象分析的若干基本步骤，分别是识别参与者、细化用例描述、合并需求获得用例和调整用例模型，需要记忆。

1.2.2.1. 用例图的元素 (重点★★★★★)

结构元素	图形符号	关系元素	图形符号
用例(Use Case)	(Use Case)	关联(Association)	——
参与者(Actor)	Actor	扩展(Extend)	<<extend>>
系统边界(System Boundary)	System	包含(Include)	<<include>>
注释(Note)	Note	泛化(Generalization)	——→
		注释链接(Note Link)	-----

用例是一种描述系统需求的方法，使用例的方法来描述系统需求的过程就是用例建模。不考虑通信关联的类型，用例图可以简化成参与者、用例和通信关联三种元素组合，如图所示。这是做题的基础，案例不会直接让你默写这些图示的意义。



1.2.2.2. 识别参与者 (重点★★★★★)

参与者是与系统交互的所有事物，该角色不仅可以由人承担，还可以是其他系统和硬件设备，甚至是系统时钟。案例真题让你分析题目中哪些属于参与者，这里要记住处理人，一些系统和设备都是。要注意的是，参与者一定在系统之外，不是系统的一部分。下面是三个具体的参与者识别的场景。

参与者类型	描述	示例
其他系统	当系统需要与其他系统交互时，该其他系统为参与者	对某企业在线教育平台系统，该企业的 OA 系统是参与者
硬件设备	若系统需与硬件设备交互，此硬件设备为参与者	开发集成电路 (IC) 卡门禁系统时，IC 卡读写器是参与者
时钟	当系统需定时触发时，时钟作为参与者	开发在线测试系统“定时交卷”功能时，引

		入时钟作为参与者
--	--	----------

1.2.2.3. 细化用例描述 (次重点★★★☆☆)

这一章节几乎没有考过，可以忽略，了解一下用例描述包含哪些部分即可。

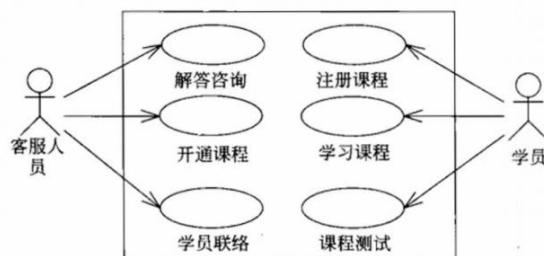
用例描述部分	内容说明
用例名称	应与用例图相符，并标注相应编号
简要说明	用简洁自然语言，描述用例为参与者传递的价值结果
事件流	参与者和系统为达成目标所发生的一系列活动
非功能需求	单列描述用例涉及的非功能需求，需保证可度量和可验证
前置条件和后置条件	前置条件为执行用例前系统必须存在的状态，不满足则用例无法启动；后置条件是用例执行完毕系统可能处于的一组状态
扩展点	若有扩展（或包含）用例，写出其名称及使用情况
优先级	表明用户对该用例的期望，确定开发先后顺序

比如我们可以这样描述注册和登录的用例。

用例描述部分	内容说明
用例名称	注册/登录 (UC01)
简要说明	为学员/老师/管理员提供安全便捷的账号接入，确保个性化学习与权限管控。
事件流	参与者输入手机号/邮箱或使用三方登录； 系统执行身份验证（密码/验证码/单点登录）； 成功后创建/加载用户会话与角色信息。
非功能需求	平均响应时间 $\leq 2s$ ；；密码与令牌加密存储 (PBKDF2/BCrypt)；传输 TLS1.2+。
前置条件和后置条件	前置条件为系统可用，认证服务正常；后置条件为会话有效并带角色与权限。
扩展点	异常重试、找回密码、二次验证(2FA)
优先级	高

1.2.2.4. 合并需求获得用例 (次重点★★★★☆)

将识别到的参与者和合并生成的用例，实际上就是合并同类项或者对于同一参与者的用例进行合并汇总。通过用例图的形式整理出来，以获得用例模型的框架，如图所示。



1.2.2.5. 调整用例模型 (重点★★★★★)

这个章节是必须要掌握的，死记硬背也要背下来，包括箭头形状，箭头指向含义，各个关系的内涵。用例之间的关系主要有包含、扩展和泛化，利用这些关系，把一些公共的信息抽取出来，以便于复用，使得用例模型更易于维护。

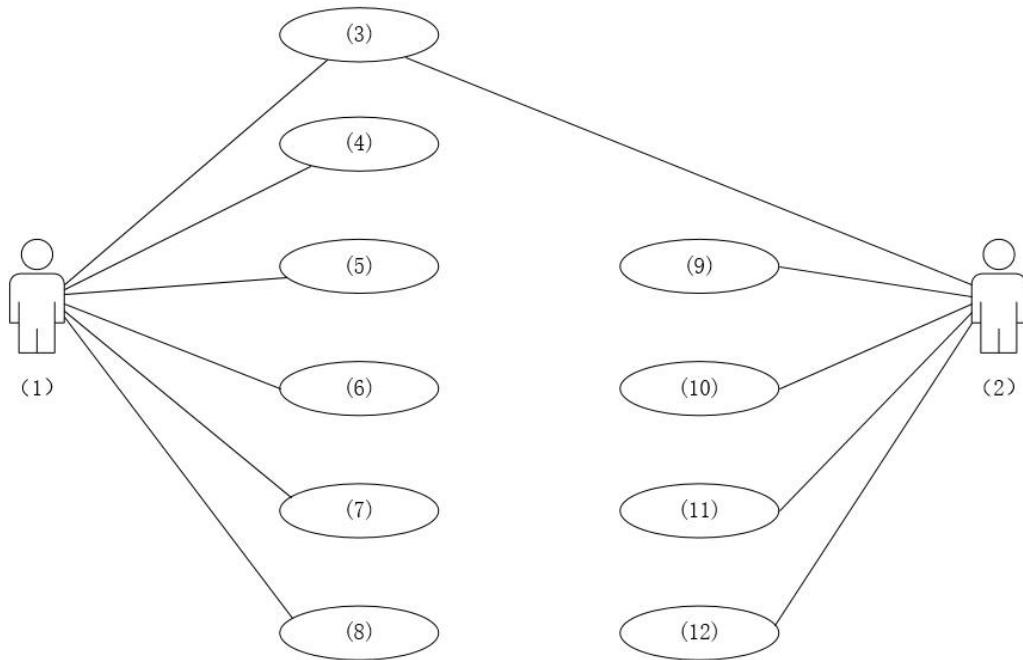
关系类型	描述	示例	构造型	箭头指向	图例
包含关系	从两个或多个用例提取公共行为，提取出的公用例为抽象用例，原始用例为基本用例。	“学习课程”和“课程测试”都需检查学员权限，“检查权限”为抽象用例，“学习课程”和“课程测试”为基本用例。	<<include>>	指向抽象用例	
扩展关系	一个用例混合多种不同场景，可分为一个基本用例和多个扩展用例。	学员“课程测试”时，若测试次数超出限额需“充入学习币”，“课程测试”是基本用例，“充入学习币”是扩展用例。	<<extend>>	指向基本用例	

泛化关系	<p>多个用例有类似结构和行为，将共性抽象为父用例，其他为子用例，子用例继承父用例所有结构、行为和关系。</p>	<p>学员课程注册可通过电话或网上注册，“注册课程”是“电话注册”和“网上注册”的父用例。</p>	→	<p>指向父用例</p> <pre> graph LR Actor((Actor)) --> CR([课程注册]) CR --> TR([电话注册]) CR --> NR([网络注册]) </pre>
------	--	---	---	---

1.2.2.6. 典型例题（重点★★★★★）

(2021 年架构真题) 某医院拟委托软件公司开发一套预约挂号管理系统，以便为患者提供更好的就医体验，为医院提供更加科学的预约管理。本系统的主要功能描述如下：(a) 注册登录，(b) 信息浏览，(c) 账号管理，(d) 预约挂号，(e) 查询与取消预约，(F) 号源管理，(g) 报告查询，(h) 预约管理，(i) 报表管理和(j) 信用管理等。

若采用面向对象方法对预约挂号管理系统进行分析，得到如图所示的用例图。请将合适的参与者名称填入图中的(1)和(2)处，使用题干给出的功能描述(a)~(j)，完善用例(3)~(12)的名称，将正确答案填在答题纸上。



这是一道套用例图壳的选词填空题。这里涉及到两个参与者，分别是患者和医生/医院管理人员/医院方都可以，两边可以看到他们对应的用例数量是不一样的。并且他们都有一个公共的用例。所以我们首先要把这用例进行分类。

患者作为预约挂号服务的直接使用者，用例围绕挂号流程展开，包括使用系统的前置操作（注册登录、信息浏览、账号管理），挂号核心操作（预约挂号、查询与取消预约），以及附加医疗服务需求（报告查询）。

医院方负责系统的运营与管理，聚焦对挂号资源及系统规则的管控。如号源管理（维护可挂号资源）、预约管理（制定预约规则与分配），以及报表管理、信用管理（从数据和信用维度优化系统运营）。

这里有一个最大的问题，就是账号管理到底属于谁的用例。假如考虑到医院方可以新增医生账号，那么属于医院方的用例，假如说账号管理包括修改信息，修改绑定手机号等功能，则属于患者的用例。这道题出得非常没水平，大家了解就行。

答案：(1) 预约人员（患者） (2) 医院管理人员 (3) (a) 注册登录

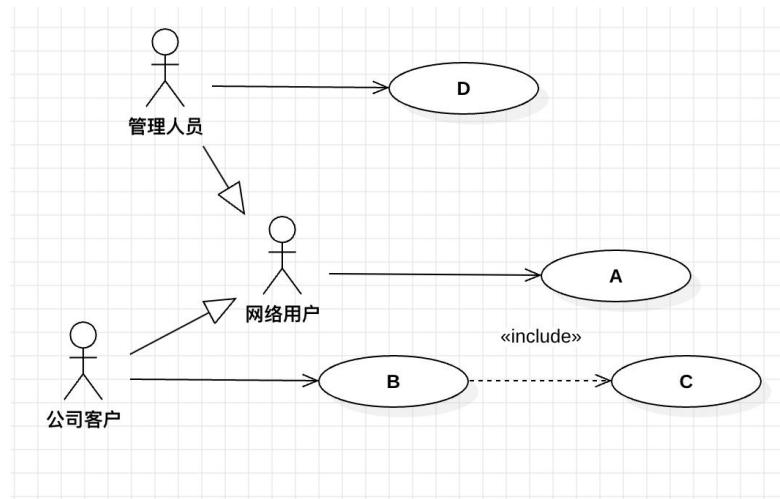
(3) - (8) (b) 信息浏览 (c) 账号管理 (d) 预约挂号 (e) 查询与取消预约 (g) 报告

查询 (9) - (12) (f) 号源管理 (h) 预约管理 (i) 报表管理 (j) 信用管理

(凯恩模拟)某电话公司决定开发一个管理所有客户信息的交互式网络系统。系统的功能如下：

1. 浏览客户信息：任何使用 Internet 的网络用户都可以浏览电话公司所有的客户信息（包括姓名、住址、电话号码等）。
2. 登录：电话公司授予每个客户一个帐号。拥有授权帐号的客户，可以使用系统提供的页面设置个人密码，并使用该帐号和密码向系统注册。
3. 修改个人信息：客户向系统注册后，可以发送电子邮件或者使用系统提供的页面，对个人信息进行修改。
4. 删除客户信息：只有公司的管理人员才能删除不再接受公司服务的客户的信息。

问：在需求分析阶段，采用 UML 的用例图描述系统功能需求，如图所示。请指出图中的 A、B、C 和 D 分别是哪个用例？



从题干给出的系统功能需求与 UML 用例图对应关系可以逐一分析：

A (浏览客户信息) : 图中 A 对应的是网络用户能够直接使用的功能，网络用户也是公司客户和管理人员参与者的父参与者，它有他们共同的功能。根据描述，任何 Internet 用户都可以浏览客户的基本信息，因此 A 就是“浏览客户信息”。该功能的参与者是“网络用户”，

与题干完全对应。

B (登录)：图中 B 与“公司客户”相关，并且用例图中有 include 关系，说明它是修改个人信息前必须执行的步骤。只有“公司客户”在系统中注册（登录）后，才能进行后续的个人信息修改，因此 B 就是“登录”。

C(修改个人信息)：客户在登录之后，才可以对个人信息进行修改。题干中明确提到“客户向系统注册后，可以通过邮件或系统页面修改个人信息”。因此 C 对应“修改个人信息”。

D (删除客户信息)：图中 D 对应的参与者是“管理员”，且题干指出“只有公司的管理员才能删除客户信息”。因此 D 就是“删除客户信息”。

答：A：浏览客户信息，B：登录，C：修改个人信息，D：删除客户信息

1.2.3.类图（重点★★★★★）

类图可以用于面向对象分析也可以用于设计，软考案例中它只会考类之间的六大关系。

1.2.3.1.类关系（重点★★★★★）

类之间的主要关系有关联、依赖、泛化、聚合、组合和实现等六大关系，它们在 UML 中的表示方式如图所示。这些关系的强弱顺序不同，下面的他们关联强度的排序结果为：泛化=实现>组合>聚合>关联>依赖，需要记忆。

这个顺序需要记忆，口诀为饭（泛）食（实现）组聚关羽（依）。（吃饭小组相聚和关羽）。

类关系一般不会让你之间默写概念，而是和特定的场景结合，让你根据实际情况建立类之间的关联模型，考察你的运用能力。

类图实际上 UML 里比较复杂的一个图，一般来说掌握这 6 个关系就已经非常不错，深挖没有必要，系分考生下表都需要记忆。其中，聚合和组合关系都是关联关系的一种。

关系类型	定义	举例	图例
关联关系	一种强语义联系的结构关系，表明两个事物间存在明确、稳定的语义联系。	工人指向任务，表明工人与任务两者存在明确关联	<pre> classDiagram class Worker { + task : Task } class Task Worker "2" --> "1" Task </pre>
依赖关系	两个类 A 和 B，若 B 的变化可能引起 A 的变化，A 依赖于 B	司机类指向汽车类，表示司机类依赖于汽车类。即司机要执行驾驶行为，必须依赖于汽车这个对象，没有汽车，司机就无法完成驾驶操作。	<pre> classDiagram class Driver { + Drive(Car car) } class Car Driver "2" *--> "1" Car </pre>
泛化关系	描述一般事物与特殊种类的关系，即父类与子类的关系	狗和猫是动物的子类，继承动物属性和方法并扩展特有功能	<pre> classDiagram class Animal class Dog class Cat Animal < -- Dog Animal < -- Cat </pre>
聚合关系	表示类之间整体与部分的关系，“部分”可同时属于多个“整体”，“部分”与“整体”生命周期可不同，好聚好散，部分能单独存在	学生和班级，学生可以脱离班级存在。	<pre> classDiagram class Student class Class Class "2" *--> "1" Student </pre>
组合关系	同样表示类之间整体与部分的关系，“部分”只能属于一个“整体”，“部分”与“整体”生命周期相同，与聚合比，生命周期同步	汽车与发动机，发动机仅属于一辆汽车，发动机随汽车创建与消亡	<pre> classDiagram class Engine class Car Engine "2" *--> "1" Car </pre>
实现关系	将说明和实践联系起来，接口说明行为，类实现接口操作	音频播放器实现“播放器”接口	<pre> classDiagram interface IPlayer { + Start() + Stop() } class AudioPlayer { + Start() + Stop() } AudioPlayer --> IPlayer </pre>

1.2.3.2. 典型例题（重点★★★★★）

(16 年架构真题) 某软件公司计划开发一套教学管理系统，用于为高校提供教学管理服务。

该教学管理系统基本的需求包括：

- (1) 系统用户必须成功登录到系统后才能使用系统的各项功能服务；
- (2) 管理员 (Registrar) 使用该系统管理学校 (University)、系 (Department)、教师 (Lecturer)、学生 (Student) 和课程 (Course) 等教学基础信息；
- (3) 学生使用系统选择并注册课程，必须通过所选课程的考试才能获得学分；如果考试不及格，必须参加补考，通过后才能获得课程学分；
- (4) 教师使用该系统选择所要教的课程，并从系统获得选择该课程的学生名单；
- (5) 管理员使用系统生成课程课表，维护系统所需的有关课程、学生和教师的信息；
- (6) 每个月到了月底系统会通过打印机打印学生的考勤信息。

项目组经过分析和讨论，决定采用面向对象开发技术对系统各项需求建模。

问：类图主要用来描述系统的静态结构，是组件图和配置图的基础。请指出在面向对象系统建模中，类之间的关系有哪几种类型？对题目所述教学服务系统的需求建模时，类 University 与类 Student 之间、类 University 和类 Department 之间、类 Student 和类 Course 之间的关系分别属于哪种类型？

第一个小问就是直接默写概念不多说。第二小问，我们首先来看，在题目要求中，类 University 与类 Student 之间的关系是整体与部分关系。并且我要换成其他业务系统的时候，不设计类 University，类 Student 可以单独设计，他们没有紧密关系，也就是说的具有不同的生存周期，口诀好聚好散，所以是聚合关系。

类 University 和类 Department 之间的关系是整体与部分的关系，两者具有相同的生存周期，你不能说设计一个系统，类 Department 但是没有类 University，这个说不通。所以它们之间是

组合关系。同理，类 Student 和类 Course 我认为也是聚合关系，但是官方是关联关系，这个说不太通。因为关联包含了聚合和组合，粒度太粗了。

答：类之间的关系包括：泛化=实现>组合>聚合>关联>依赖。类 University 与类 Student 之间的关系是聚合关系。类 University 与类 Department 之间的关系是组合关系。类 Student 与类 Course 之间的关系是关联关系。

(凯恩模拟) 类通常不会单独存在，因此当对系统建模时，不仅要识别出类，还必须对类之间的相互关系建模。在面向对象建模中，提供了 4 种关系：依赖 (dependency) 、概括 (generalization) 、关联 (association) 和聚合 (aggregation) 。

问：请说明关联和聚合之间的主要区别。

在 UML 类图中，关联与聚合都属于类与类之间的结构性关系，但其语义范围不同：

关联 (Association) : 描述类之间的联系，侧重于“知道”或“使用”。例如，学生和课程之间存在“选修”关系，学生对象可以访问课程对象。关联关系可以是双向的（默认），也可以指定为单向。关联常用于描述对象之间的业务逻辑联系。

聚合 (Aggregation) : 是关联的特化，描述整体与部分的关系。例如，“计算机”由“显示器、键盘、鼠标”等部件组成。整体与部分可以独立存在，比如“显示器”即使没有“计算机”也可以单独使用，因此称为空心菱形的弱聚合关系。若生命周期也依赖整体（如“房子”和“房间”），则通常进一步用 组合 (Composition, 实心菱形) 来表示。

关联 (Association) : 表示两个类之间的一种结构性关系，一个类的对象可以“知道”另一个类的对象，并能调用其方法。关联关系通常是双向的，也可以是单向的，语义上强调 对象之间的联系。

聚合 (Aggregation) : 是关联关系的一种特殊形式，表示整体与部分的关系，即“has-a”关系。它强调一个类是另一个类的组成部分，通常用 带空心菱形的线条 表示。

1.2.4.顺序图（重点★★★★★）

顺序图和活动图是案例常考的内容，放在这里作为补充。特别是它们各自对应的元素、图形符号都要了解含义。顺序图是一种最常用的动态交互图，用于显示对象间的交互活动，关注对象之间消息传送的时间顺序。在软考案例中直接考察概念只有一次（组合片段），多数是简答填空的形式让你对业务场景进行建模。

1.2.4.1.顺序图的元素（重点★★★★★）

类别	详情
核心概念	对象、生命线、执行发生、消息，交互片段
推荐使用场合	用例分析、用例设计等场合

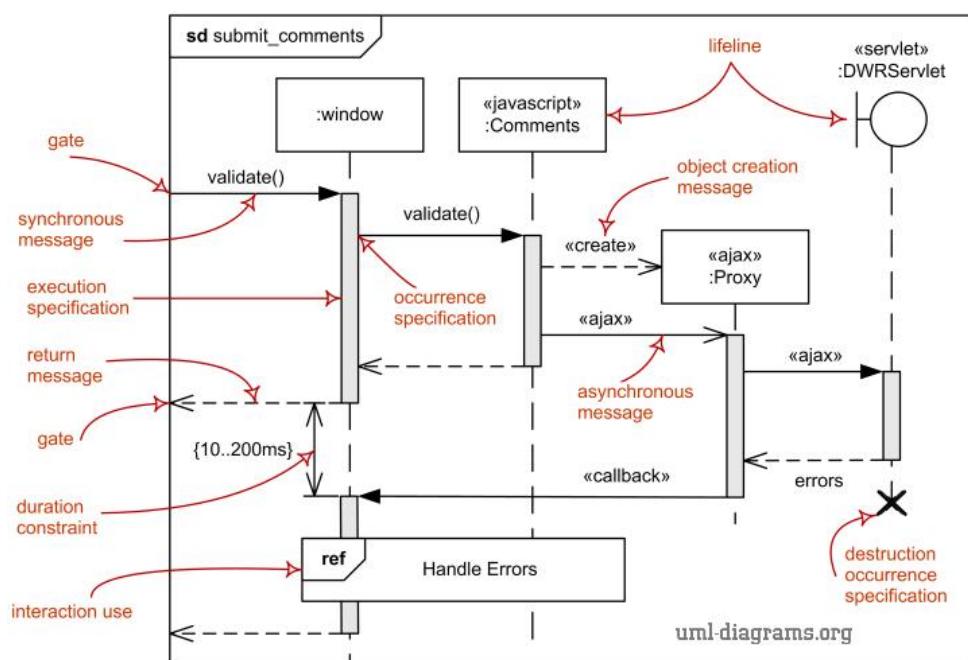
元素	图形符号	元素	图形符号
对象/生命线 (Object/Lifeline)		同步消息 (Synchronous Message)	
交互片段 (Interaction Frame)		异步消息 (Asynchronous Message)	
执行发生 (Execution Occurrence)		返回消息 (Return Message)	
状态不变式 (State Invariant)		创建消息 (Create Message)	

概念	解析
Lifeline (生命线)	代表对象或参与者生命周期的垂直虚线，贯穿其存在时间段，直观呈现对象在交互中的存在周期。

Message (消息)	对象间交互的载体，通过箭头表示，包含同步（实心三角箭头）、异步（开放箭头）等类型，标注消息名称或操作，驱动交互行为。
Endpoint (端点)	位于生命线底部的终止标记，用于标识对象生命周期结束（如对象销毁），明确对象存在的起止边界。
Gate (门)	连接不同交互片段（如组合片段）的接口，标记消息流入/流出点，在复杂交互中管理消息流向，支持片段嵌套或引用。
CombinedFragment(组合片段)	容纳多个交互操作数的容器，用于描述复杂逻辑。常见类型：alt（条件分支）、loop（循环）、par（并行）、opt（可选）等

CombinedFragment（组合片段，也可叫作交互片段），上图和上表表述不一致这里提醒一下。

典型的顺序图如下所示。



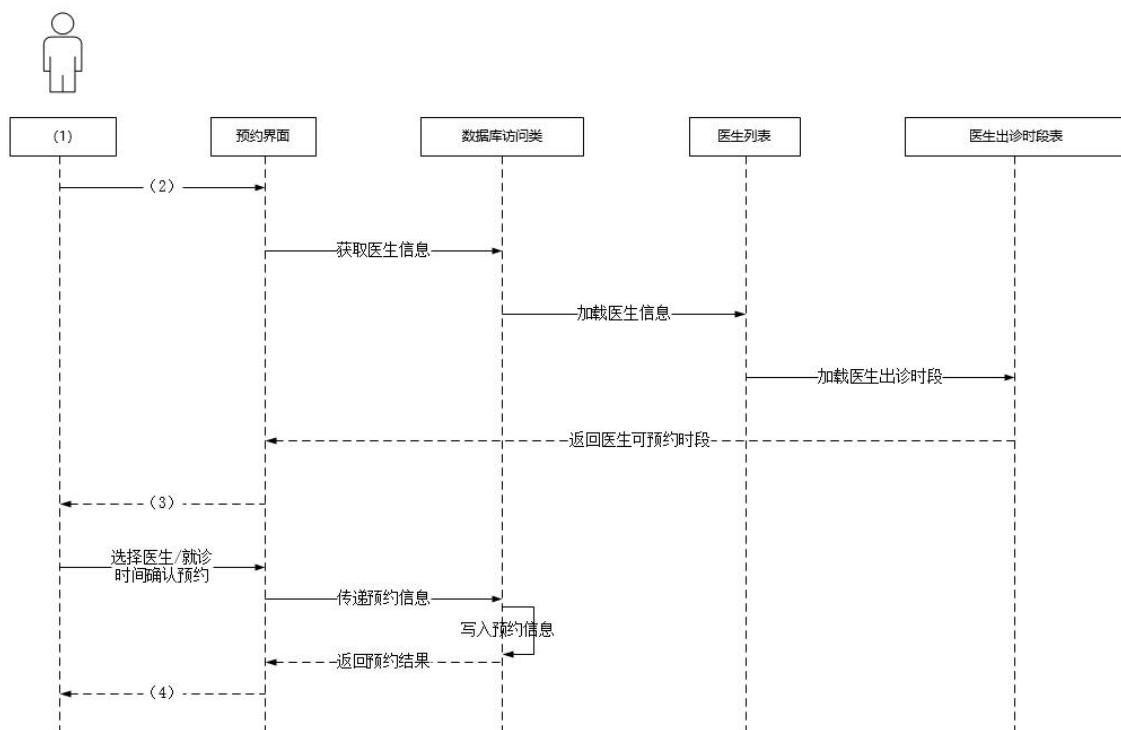
1.2.4.2. 典型例题（重点★★★★★）

(21 年架构真题) 某医院拟委托软件公司开发一套预约挂号管理系统，以便为患者提供更好的就医体验，为医院提供更加科学的预约管理。本系统的主要功能描述如下：(a) 注册登录，(b) 信息浏览，(c) 账号管理，(d) 预约挂号，(e) 查询与取消预约，(F) 号源管理，

(g) 报告查询, (h) 预约管理, (i) 报表管理和(j) 信用管理等。

问：预约人员（患者）登录系统后发起预约挂号请求，进入预约界面。进行预约挂号时使用数据库访问类获取医生的相关信息，在数据库中调用医生列表，并调取医生出诊时段表，将医生出诊时段反馈到预约界面，并显示给预约人员；预约人员选择医生及就诊时间后确认预约，系统反馈预约结果，并向用户显示是否预约成功。

采用面向对象方法对预约挂号过程进行分析，得到如图所示的顺序图，使用题干中给出的描述，完善图中对象（1），及消息（2）~(4) 的名称，将正确答案填在答题纸上。



该问考查 UML 中的顺序图，紧扣题目描述来组织内容即可，从题干中“预约人员（患者）登录系统后发起预约挂号请求，进入预约界面”的信息可知（1）应为预约人员（患者）（2）为预约挂号请求；从题干中“将医生出诊时段反馈到预约界面，并显示给预约人员”的信息可知（3）应为显示医生可预约时段；从题干中“系统反馈预约结果，并向用户显示是否预约成功”的信息可知（4）应为显示预约是否成功。

答案：（1）预约人员（患者）（2）预约挂号请求（3）显示医生可预约时段（4）显示预

约是否成功

1.2.5.通信图/协作图（重点★★★★★）

通信图/协作图是 UML 交互图的一种，在 UML1.x 版本中被称为协作图。一个通信图显示了对象间的联系以及对象间发送和接收的消息。它和顺序图的最大区别就是它不反映时序信息。

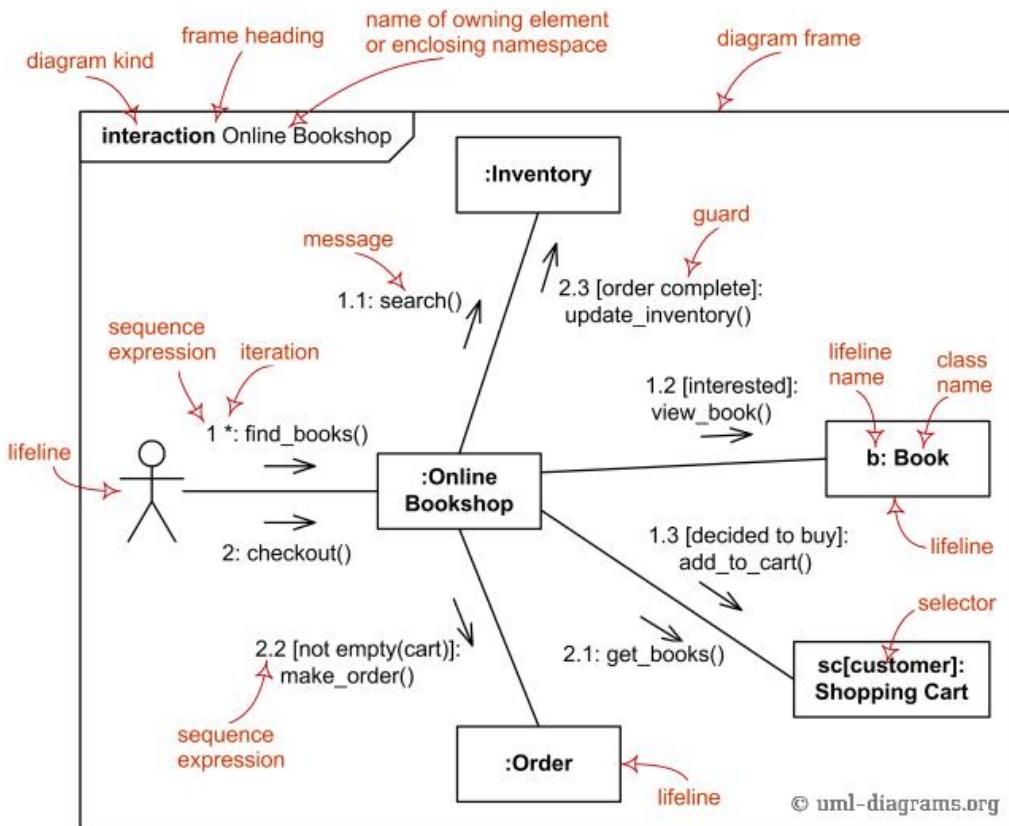
1.2.5.1.通信图/协作图的元素（重点★★★★★）

通信图常用的建模元素如下所示：包括对象（Object）、通信链接（Link）、消息（Message）。

概念	解析
对象	类的实例，也可代表协作、组件、节点等事物的实例
通信链接	对象间发送消息的路径，是通信图特有的连接元素
消息	对象间发送和接收的交互内容，体现动态行为

元 素	图形符号	元 素	图形符号
对象/生命线 (Object/Lifeline)	Object: Class	同步消息 (Synchronous Message)	→
链接(Link)	——	异步消息 (Asynchronous Message)	→
		返回消息 (Return Message)	←----

这里的同步消息、异步消息和返回消息的箭头和顺序图是一样的。典型的通信图如下所示：



1.2.5.2. 典型例题（重点★★★★★）

(21 年架构真题) 请简要说明在描述对象之间的动态交互关系时, 协作图与顺序图存在哪些区别。

答: 协作图与顺序图侧重点不同: 顺序图以时间轴为核心, 通过垂直生命线和消息箭头清晰展示消息传递的时序关系, 适合体现复杂流程、并发或异步操作; 协作图则以对象间的结构关系为中心, 通过节点和链接呈现交互路径, 强调对象协作的拓扑结构, 适合分析静态关系紧密、消息路径简单的场景。二者语义等价可相互转换, 实际应用中需根据需求选择——关注时序细节用顺序图, 强调结构关系用协作图, 也可结合使用以全面描述交互逻辑。

1.2.6. 活动图 (重点★★★★★)

将业务流程或其他计算的结构展示为内部一步步的控制流和数据流, 主要用于描述某一方

法、机制或用例的内部行为，适用于业务建模、需求、类设计等场合。它的作用和流程图、数据流图、状态图非常相似，常被拿来一起比较。

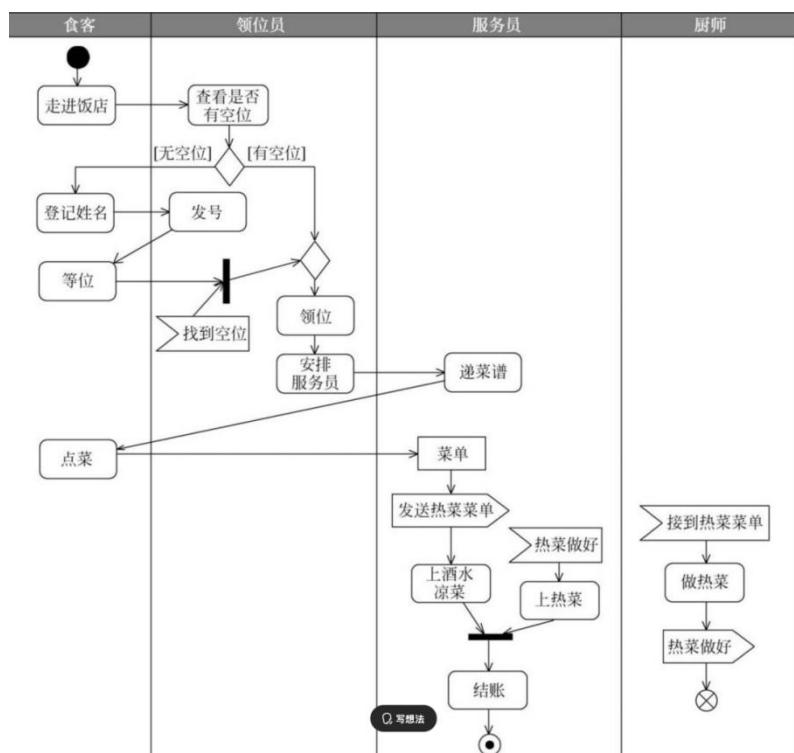
1.2.6.1. 活动图的元素 (重点★★★★★)

活动图和我们传统的流程图最大的区别在于它可以做并行（通过分叉/合并）的表示，同时还有事件发送和接收系统，相当于他可以做一个异步的表达，这是最大的不同。

概念	解析
初始节点	活动的起点，表示流程开始
终止节点	活动的终点，表示流程结束
动作状态	原子级操作或活动步骤
决策节点	用于流程分支，根据条件选择路径
合并节点	汇合多个分支流
分叉/合并	并行处理多个控制流（分叉）及汇合（合并）
泳道（分区）	按角色/部门划分的垂直区域，明确动作执行主体
对象节点	表示对象状态变化或数据流
控制流	连接动作节点的有向线段，表示执行顺序
对象流	连接对象节点的有向线段，表示对象传递
信号节点	异步通信的发送/接收事件

元 素	图形符号	元 素	图形符号
活动/动作(Activity/Action)	(Activity)	起点(InitialNode)	●
对象(Object)	Object	终点(FinalNode)	◎
发送事件(SendEvent)	Event	流结束(FlowFinal)	⊗
接收事件(AcceptEvent)	>Event	分叉/合并(Fork/Join)	——
分区(Partition)	——	控制流(ControlFlow)	——→
决策点(Decision)	◇	对象流(ObjectFlow) (在 UML 1.x 中为虚线)	——→

活用活动图是关键，大家看下图这个例子。这里体现了分叉合并和决策、发送接收事件等关键用法，把下面的解析一定要看懂。



这里列出了一个相对完整的活动图，该活动图描述了饭店业务中食客吃饭业务流程。图中首先通过分区机制将整个活动图划分为 4 个区域（食客、领位员、服务员和厨师），名称代表该区域的动作主体（即由指定的角色执行该区域的动作）。

具体的业务流程如下。首先，从起点开始，通过控制流进入第一个动，即食客走进饭店，饭店领位员帮忙查看是否有空位。该动作执行完成后，会产生有无空位的分支，通过决策节点

描述，两种分支情况分别放置在决策节点的输出控制流上。

如果有空位则直接领位，进入后续动作；反之，如果无空位，则食客登记姓名后，领位员发号表明等待的顺序，食客等位。在等位期间，领位员会频繁检查是否有空位（图中没有建模该动作），如果有空位，则发送找到空位的信号给对应的食客（这是一个领位员执行的独立控制流），这时这两个控制流汇合，并通过合并节点与有空位的分支流合并进入领位动作。

领位完成后，领位员安排服务员点菜，服务员递菜谱，食客根据菜谱点菜。点菜完成后，通过对象流形成菜单对象，服务员将菜单中的热菜发送给厨师（异步发送事件），从而通知厨师做热菜，之后安排上酒水凉菜。厨师接到热菜菜单后（异步接收事件），即启动独立地做热菜流程，热菜做好后发送信号通知服务员上热菜，当前厨师的控制流结束。服务员接收到热菜做好的信号后，为食客上主菜，这是与之前上酒水凉菜的控制流相对独立的一个新流程。

在酒水凉菜和主菜都上完后，汇合在一起进入后续的结账环节（省略了中间的食客吃饭的过程）。结完账后，进入终点表明整个流程结束。

1.2.6.2. 典型例题（重点★★★★★）

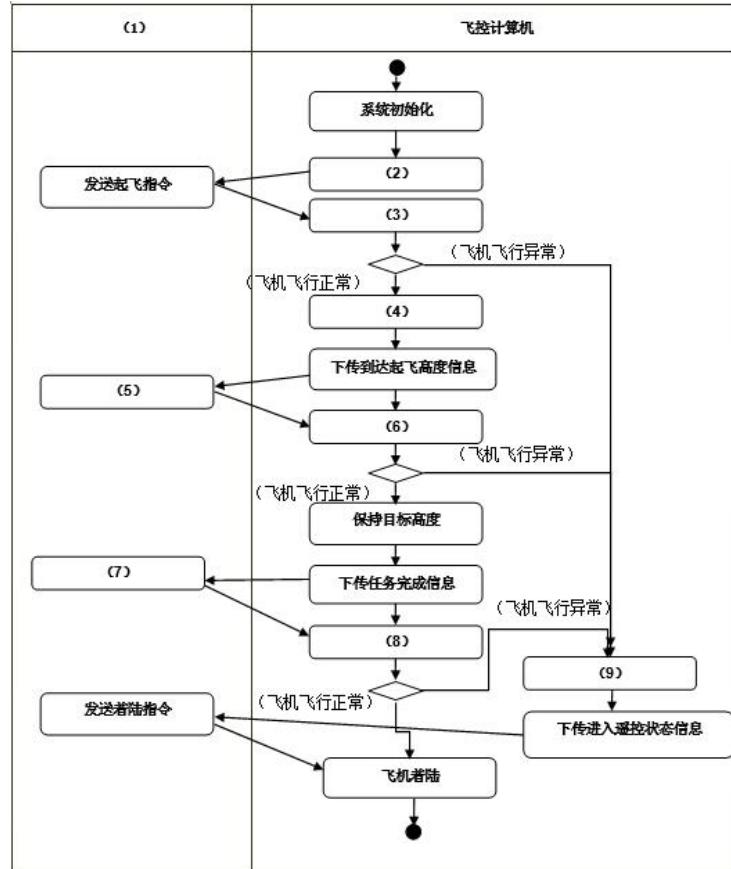
(15 年架构真题) 某公司拟研制一款高空监视无人直升机，该无人机采用遥控—自主复合型控制实现垂直升降。该直升机飞行控制系统由机上部分和地面部分组成，机上部分主要包括无线电传输设备、飞控计算机、导航设备等，地面部分包括遥控操纵设备、无线电传输设备以及地面综合控制计算机等。其主要工作原理是地面综合控制计算机负责发送相应指令，飞控计算机按照预定程序实现相应功能。经过需求分析，对该无人直升机控制系统纵向控制基本功能整理如下：

- (a) 飞控计算机加电后，应完成系统初始化，飞机进入准备起飞状态；
- (b) 在准备起飞状态中等待地面综合控制计算机发送起飞指令，飞控计算机接收到起飞指令

后，进入垂直起飞状态；

- (c) 垂直起飞过程中如果飞控计算机发现飞机飞行异常，飞行控制系统应转入无线电遥控飞行状态，地面综合控制计算机发送遥控指令；
- (d) 垂直起飞达到预定起飞高度后，飞机应进入高度保持状态；
- (e) 飞控计算机在收到地面综合控制计算机发送的目标高度后，飞机应进入垂直升降状态，接近目标高度；垂直升降过程中出现飞机飞行异常，控制系统应转入无线电遥控飞行；
- (f) 飞机到达目标高度后，应进入高度保持状态，完成相应的任务；
- (g) 飞机在接到地面综合控制计算机发送的任务执行结束指令后，进入飞机降落状态；
- (h) 飞机降落过程中如果出现飞机飞行异常，控制系统应转入无线电遥控飞行；
- (i) 飞机降落到指定着陆高度后，进入飞机着陆状态，应按照预定着陆算法，进行着陆；
- (j) 无线电遥控飞行中，地面综合控制计算机发送着陆指令，飞机进入着陆状态，应按照预定着陆算法，进行着陆。

问：根据题目中描述的基本功能需求，架构师王工给出了无人直升机控制系统纵向控制的顶层活动图。请根据题干描述，完善图活动图的(1)-(9)，将答案填写在答题纸中



此题看似考察活动图，实际上考察你业务需求理解的能力。

(1) 地面综合控制计算机。题干中明确提到“地面综合控制计算机负责发送相应指令，飞控计算机按照预定程序实现相应功能”，在整个活动图中，与飞控计算机交互并发送指令的地面部分设备就是地面综合控制计算机，所以(1)处应填地面综合控制计算机。

(2) 下传起飞就绪信息。根据需求描述“飞控计算机加电后，应完成系统初始化，飞机进入准备起飞状态；在准备起飞状态中等待地面综合控制计算机发送起飞指令”，在进入准备起飞状态后，飞控计算机需要向地面综合控制计算机反馈一个状态信息，告知其已准备好，所以这里(2)是下传起飞就绪信息，以便地面综合控制计算机决定是否发送起飞指令。

(3) 垂直起飞。由“飞控计算机接收到起飞指令后，进入垂直起飞状态”可知，当飞控计算机接收到来自地面综合控制计算机的起飞指令后，下一步动作就是进入垂直起飞状态，所以(3)为垂直起飞。

(4) 高度保持。依据“垂直起飞达到预定起飞高度后，飞机应进入高度保持状态”，在垂直起飞完成并达到预定高度这个条件满足后，飞机接下来应进入的状态就是高度保持，故(4)填高度保持。

(5) 发送目标高度。从“飞控计算机在收到地面综合控制计算机发送的目标高度后，飞机应进入垂直升降状态”可知，在高度保持之后，地面综合控制计算机要向飞控计算机发送一个新的指令信息，即目标高度，从而让飞机进入下一阶段，所以(5)是发送目标高度。

(6) 垂直升降。按照“飞控计算机在收到地面综合控制计算机发送的目标高度后，飞机应进入垂直升降状态”，当接收到目标高度指令后，飞机对应的动作状态就是垂直升降，因此(6)为垂直升降。

(7) 发送任务结束指令。根据“飞机在接到地面综合控制计算机发送的任务执行结束指令后，进入飞机降落状态”，在飞机完成相应任务后，地面综合控制计算机需要发送一个指令来让飞机进入降落阶段，这个指令就是任务结束指令，所以(7)是发送任务结束指令。

(8) 飞机降落。由“飞机在接到地面综合控制计算机发送的任务执行结束指令后，进入飞机降落状态”可知，接收到任务结束指令后，飞机进入的状态是飞机降落状态，故(8)填飞机降落。

(9) 无线电遥控飞行。从“垂直起飞过程中如果飞控计算机发现飞机飞行异常，飞行控制系统应转入无线电遥控飞行状态”“垂直升降过程中出现飞机飞行异常，控制系统应转入无线电遥控飞行”“飞机降落过程中如果出现飞机飞行异常，控制系统应转入无线电遥控飞行”可知，当飞行出现异常时，控制系统会转入的状态是无线电遥控飞行，所以(9)为无线电遥控飞行。

答：(1) 地面综合控制计算机 (2) 下传起飞就绪信息 (3) 垂直起飞 (4) 高度保持 (5) 发送目标高度 (6) 垂直升降 (7) 发送任务结束指令 (8) 飞机降落 (9) 无线电遥控飞行

1.2.7. 常见图对比 (重点★★★★★)

讲了那么多图，实际上让你说对比也不太容易。为此，凯恩整理了历年真题中出现过的各个图之间概念的比较，学习的时候要重点关注对比结论，实际上对比其实就是罗列他们的异同，把各自概念写出来，异同自然而然可以对比出来。

图类型	主要关注点	典型用途	对比结论
用例图(UseCaseDiagram)	系统外部用户(参与者)与系统之间的交互	捕获需求、确定系统边界、展示用户价值	抽象需求视角，描述“做什么”；不涉及内部实现
类图(ClassDiagram)	系统内部类、属性、方法及类之间的关系	定义系统的结构设计、支撑代码实现	实现视角，描述“怎么做”；和用例图形成需求→设计的映射
数据流图(DFD)	数据的加工、流动、存储过程	需求分析阶段，建模系统数据处理逻辑	关注“数据在系统中如何流动”，忽略具体实现细节
活动图(ActivityDiagram)	工作流、活动之间的顺序、分支、并行	表达业务流程或算法步骤	更偏向“任务执行步骤”，强调行为和流程控制
流程图(Flowchart)	操作步骤及逻辑判断	表达算法逻辑、简单过程	与活动图类似，但更传统，常用于说明程序逻辑
状态图(StateMachineDiagram)	对象的状态变化及触发条件	建模复杂对象的生命周期	关注“对象随事件演化的状态”，区别于活动图的任务流
顺序图(SequenceDiagram)	对象间消息的时间顺序	展示场景交互流程，强调消息调用先后	强调“时序”，适合表现请求响应的详细过程
通信图(CommunicationDiagram)	对象之间的连接关系及消息传递	表达协作结构和对象间关系	强调“结构与协作”，和顺序图互补：内容相同，视角不同

2. 上篇- 系统设计/面向对象设计 (重点★★★★★)

架构、系分里的系统设计类考题只考面向对象设计方面内容，结构化设计基本不涉及，所以这里的标题取成系统设计/面向对象设计。这里主要考察两块内容，第一是类识别，一般来说会

给你场景让你判断，某个类是边界类，控制类，还是实体类，这个方面架构考的少，系分考的多。第二类考点是类设计原则，架构案例还没考过，系分已经考过。一般会以概念题的形态出现，纯八股，需要好好记忆。

2.1.类识别（次重点★★★☆☆）

2.1.1.识别原则（次重点★★★☆☆）

一个类可以有多种职责，设计得好的类一般至少有一种职责，在定义类时，将类的职责分解为类的属性和方法，其中属性用于封装数据，方法用于封装行为。在系统设计过程中，类可以分为三种类型：实体类、边界类和控制类。它们的意义如下所示。我们重点关注类识别方法，也就是词性法。这里要强调的是边界类比较特殊，我们一般把窗口、通信协议、打印机接口、报表等内容当作边界类，其他都做实体类。

类型	识别方法（词性）	作用	示例
实体类	通常是名词	保存需永久存储的信息，属性和关系	在线教育平台系统的学员类、课程类
控制类	由动宾结构的短语（“动词+名词”或“名词+动词”）纯动词也可以	用于控制用例工作，对一个或几个用例特有的控制行为建模行为具有协调性	用例“身份验证”对应的控制类“身份验证器”
边界类	通常是名词（一般来说就例子所示的内容）	用于系统接口与外部交互，将系统与外部环境变更分隔开	窗口、通信协议、打印机接口、报表或其他可以连接外部和控制类的东西。

2.1.2.典型例题（重点★★★★★）

(21 年系分真题) 某高校拟开发一套图书馆管理系统，在系统分析阶段，系统分析师整理的核心业务流程与需求如下：

系统为每个读者建立一个账户，并给读者发放读者证（包含读者证号、读者姓名），账户中存储读者的个人信息、借阅信息以及预订信息等，持有读者证可以借阅图书、返还图书、查询图书信息、预订图书、取消预订等。在借阅图书时，需要输入读者所借阅的图书名、ISBN 号，然后输入读者的读者证号，完成后提交系统，以进行读者验证。如果读者有效，借阅请求被接受，系统查询读者所借阅的图书是否存在，若存在，则读者可借出图书，系统记录借阅记录；如果读者所借阅的图书已被借出，读者还可预订该图书。读者如期还书后，系统清除借阅记录，否则需缴纳罚金，读者还可以选择续借图书。同时，以上部分操作还需要系统管理员和图书管理员参与。

问：设计类图的首要工作是进行类的识别与分类，该工作可分为两个阶段：首先，采用识别与筛选法，对需求分析文档进行分析，保留系统的重要概念与属性，删除不正确或冗余的内容；其次，将识别出来的类按照边界类、实体类和控制类等三种类型进行分类。请用 200 字以内的文字对边界类、实体类和控制类的作用进行简要解释，并对下面给出的候选选项进行识别与筛选，将合适的候选选项编号填入表中的（1）～（3）空白处，完成类的识别与分类工作。

类型	实例
边界类	(1)
实体类	(2)
控制类	(3)

候选选项：a) 系统管理员 b) 图书管理员 c) 读者 d) 读者证 e) 账户 f) 图书 g) 借阅 h) 归还 i) 预订 j) 罚金 k) 续借 l) 借阅记录

第一小问纯概念不再展开，看看第二问。控制类前面说过一般是动宾结构或者动词，所以这里的 g (借阅)、h (归还)、i (预订)、k (续借) 都是控制类。边界类这里不好判断，之前凯恩说过边界类窗口、通信协议、打印机接口、报表等，好像看起来都不像。那只能从定义

判断，就是可以沟通外接和控制类的东西。这里读者证是最合适的。其余的都可以看作实体类。

答：边界类主要用于描述外部参与者与系统之间的交互。边界类是一种用于对系统外部环境与其内部运作之间的交互进行建模的类。这种交互包括转换事件，并记录系统表示方式（例如接口）中的变更。实体类主要是作为数据管理和业务逻辑处理层面上存在的类。实体类的主要职责是存储和管理系统内部的信息，它也可以有行为，甚至很复杂的行为，但这些行为必须与它所代表的实体对象密切相关。

控制类用于描述一个用例所具有的事件流控制行为，控制一个用例中的事件顺序。控制类是控制其他类工作的类。每个用例通常有一个控制类，控制用例中的事件顺序，控制类也可以在多个用例间共用。其他类通常并不向控制类发送消息，而是由控制类发出消息。

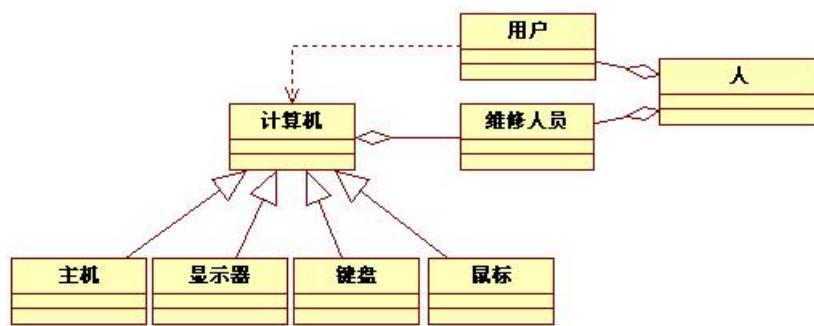
第二问官方的答案如下（说不通）

- (1) j、l (2) a、b、c、f (c 可替换成 e) (3) g、h、i、k

凯恩给出的答案是

- (1) d (2) a、b、c、e、f、j、l (3) g、h、i、k

(凯恩模拟) 找出并说明下面类图中的错误



这张类图错误的根源在于对 UML 中类关系语义的误用，主要表现在继承、聚合和关联的混淆。逐一分析如下：

计算机与部件关系问题：类图中“主机、显示器、键盘、鼠标”被建模成“计算机”的子类，这是

错误的。继承关系表示 is-a (是一种)，例如“猫是动物”，而计算机与其部件明显是 part-of (整体与部分) 关系。正确的建模方式应当使用聚合 (空心菱形)，表示计算机由这些部件构成。若要强调部件生命周期依赖计算机，可以进一步用 组合 (实心菱形)。

人、用户、维修人员关系问题

类图中使用聚合连接“人”与“用户/维修人员”，这是错误的。用户和维修人员并不是“人”的组成部分，而是“人”的不同角色。它们与“人”的关系应是 继承 (泛化)，即用户和维修人员都是人，因此应使用 实线+空心三角箭头，指向父类“人”。

维修人员与计算机关系问题

图中建模成维修人员与计算机的聚合关系，这是错误的。维修人员并不“拥有”计算机，而是对计算机进行维护，这是一种 使用关系。正确的表示方式应是 依赖关系 (虚线箭头，指向被依赖类)，或者简单的 单向关联 (实线箭头)，表示维修人员使用或维护计算机。

该类图中的主要错误如下：

1. 计算机与其部件之间不是继承关系，应是聚合关系。2. 人与用户、维修人员之间不是聚合关系，应是继承关系。3. 维修人员与计算机之间不是聚合关系，应是维修人员依赖计算机或单向关联。

2.2. 面向对象设计原则 (重点★★★★★)

在面向对象设计中，可维护性的复用是以设计原则为基础的。常用的面向对象设计原则包括单一职责原则 (S)、开闭原则、里氏替换原则、依赖倒置原则、组合/聚合复用原则、接口隔离原则和最少知识原则等。每个原则具体是什么意思需要记忆，系分案例有可能默写。

2.2.1.设计原则（重点★★★★★）

设计原则	详细描述
单一职责原则 (S)	单一职责原则要求类应该只有一个设计目的
开闭原则 (O)	可扩展系统并提供新行为，通过添加抽象层实现，是其他原则的基础。
里氏替换原则 (L)	软件实体使用基类对象时适用于子类对象，反之不一定，设计时应将变化类设计为抽象类或接口。
接口隔离原则 (I)	分逻辑和狭义两种理解，前者将接口视为角色，后者要求接口提供客户端需要的行为，将大接口方法分至小接口。
依赖倒置原则 (D)	抽象不依赖于细节，针对接口编程，是实现开闭原则的主要机制，与各种技术和框架相关。
组合/聚合复用原则	通过组合或聚合关系复用已有对象，比继承更灵活，耦合度低。
最少知识原则（迪米特法则）	软件实体应尽量少与其他实体相互作用

2.2.2.典型例题（重点★★★★★）

(24年系分真题) 某高校需开发一套在线论文管理系统，功能包括：学生用户通过界面登录、查看可选课题列表、提交选题；教师用户可发布课题、批阅论文；管理员管理用户权限；系统需支持论文上传下载、审核状态跟踪等功能。

问：请用 150 字简要说明什么是面向对象设计的开闭原则。

此题考察面向对象设计原则，这是反复让大家记忆背诵的内容。

答：面向对象设计的开闭原则（OpenClosedPrinciple, OCP）指的是一个软件实体（如类、模块、函数等）应该对扩展开放，对修改关闭。也就是说，在设计一个模块时，应当使该模块可以在不被修改源代码的前提下被扩展，即能够在不必修改原有代码的情况下改变这个模块的行为。

3.上篇-数据库系统基础（超级重点★★★★★）

数据库系统是系统分析案例师常考知识点，超级重点毫无疑问。案例常考函数依赖，规范化理论，并发控制，完整性，ER 图等。有一部分在红宝书一本全上已经有了，为了方便大家看还是放进来。

3.1.关系模型（重点★★★★★）

关系模型包含元素如下所示，这里所有的名词都要理解。倒不是会出题目，而是因为下面的术语是你理解后面进一步概念的基础。

术语	描述
关系(表文件)	一个二维表，由行和列组成，对应数据库中的一张表。
元组（记录）	表中的一行，代表一个元组或一条记录。
属性（字段）	表中的每一列，定义了数据的意义和数据类型。
属性值	行和列交叉点的值，代表特定记录的特定属性的具体数据。
主码（主键）	用于唯一确定表中一个元组的数据，可以是一个或多个字段。
域	属性的取值范围，定义了字段可以存储的数据类型和可能的值。
关系模式	关系的逻辑描述，一般表示为：关系名（属性 1, 属性 2, ..., 属性 n）
主属性	包含在任何一个候选键中的属性。
非主属性	不包含在任何候选键中的属性。

3.1.1.关系运算基础（次重点★★★☆☆）

这一章需要重点掌握。在选择题中时常考到。特别这里的关系运算符号一定要记住，有时候会给你一个关系代数让你写出等价的 SQL 表达式，所以看懂非常关键。关系代数的基本运

算主要有并、交、差、笛卡尔积、选择、投影、连接和除法运算。下面的 \equiv 是等价于的意思。

(1) 并。计算两个关系在集合理论上的并集，即给出关系 R 和 S （两者有相同元/列数）的元组包括 R 和 S 所有元组的集合，形式定义如下，式中 t 是元组变量（下同）：

$$R \cup S \equiv \{t | t \in R \vee t \in S\}$$

R			S			$R \cup S$		
A	B	C	A	B	C	A	B	C
1	1	1	5	2	0	5	2	0
1	2	1	1	2	2	1	2	2
2	3	3	1	1	1	1	1	1

(2) 差。计算两个关系区别的集合，即给出关系 R 和 S （两者有相同元/列数）， $R-S$ 的元组包括 R 中有而 S 中没有的元组的集合，形式定义如下：

$$R - S \equiv \{t | t \in R \wedge t \notin S\}$$

R	S	上面是 $R - S$ 下面是 $S - R$

A	B	C
1	1	1
1	2	1
2	3	3

A	B	C
5	2	0
1	2	2
1	1	1

A	B	C
1	2	1
2	3	3

A	B	C
5	2	0
1	2	2

(3) 交。计算两个关系集合理论上的交集，即给出关系 R 和 S (两者有相同元/列数)，

$R \cap S$ 的元组包括 R 和 S 相同元组的集合，形式定义如下：

$$R \cap S \equiv \{t | t \in R \wedge t \in S\}$$

R			S			$R \cap S$		
A	B	C	A	B	C	A	B	C
1	1	1	5	2	0	1	1	1
1	2	1	1	2	2			
2	3	3	1	1	1			

(4) 笛卡尔积。计算两个关系的笛卡尔乘积，令 R 为有 m 元的关系，S 为有 n 元的关系，则 $R \times S$ 是 $m+n$ 元的元组的集合，其前 m 个元素来自 R 的一个元组，而后 n 个元素来自 S 的一个元组。形成定义如下：

$$R \times S \equiv \{t | t = \langle t_r, t_s \rangle \wedge t_r \in R \wedge t_s \in S\}$$

若 R 有 u 个元组，S 有 v 个元组，则有 $u \times v$ 个元组。

R	S	$R \times S$
---	---	--------------

A	B	C
1	1	1
1	2	1
2	3	3

C	D	E
5	2	0
1	2	2
1	1	1

R.A	R.B	R.C	S.C	S.D	S.E
1	1	1	5	2	0
1	1	1	1	2	2
1	1	1	1	1	1
1	2	1	5	2	0
1	2	1	1	2	2
1	2	1	1	1	1
2	3	3	5	2	0
2	3	3	1	2	2
2	3	3	1	1	1

(5) 投影。从一个关系中抽取指明的属性(列)，就是 SQL 中 SELECT 指定的列名。假设有一个包含属性 A 的关系，则

$$\pi_A(R) \equiv \{t[A] | t \in R\}$$

(6) 选择。从关系中抽取出满足给定限制条件的记录，就是 SQL 中 WHERE 后面带的条件记作，其中 F 表示选择条件，是一个逻辑表达式(逻辑运算符+算术表达式)。

$$\sigma_F(R) \equiv \{t | t \in R \wedge F(t) = \text{true}\}$$

(7) θ 连接。可以把它看作是先做笛卡尔积然后筛选出满足条件的元组，如果两个关系中进行比较的分量是相同的属性组，并且在结果中将重复的属性去掉，则称为自然连接，记作：

$$R \bowtie S \equiv \{t_r, t_s | t_r \in R \wedge t_s \in S \wedge t_r[C] = t_s[C]\}$$

R	S	R \bowtie S
---	---	---------------------------------

A	B	C	C	D	E	A	B	C	D	E
1	1	1	5	2	0	1	1	1	2	2
1	2	1	1	2	2	1	1	1	1	1
2	3	3	1	1	1	1	2	1	2	2
						1	2	1	1	1

除法不展开讨论比较复杂毫无意义。考到就放弃。

3.2. 数据库设计与建模（超级重点★★★★★）

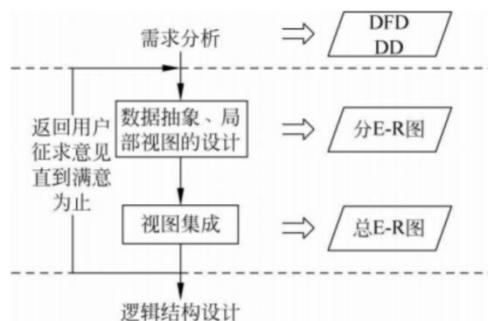
基于数据库系统生命周期的数据库设计在系分教材上可分为如下 5 个阶段：规划、需求分析、概要设计、逻辑设计和物理设计。有的教材写 6 个阶段，需求分析、概要设计、逻辑设计、物理设计、数据库实施和数据库运行和维护阶段。所以这里凯恩建议主要关注四块内容，既需求分析、概要设计、逻辑设计、物理设计阶段，其中概要设计、逻辑设计是最重要，其他不重要。

阶段	主要内容	具体任务
需求分析	设计者需了解和分析用户需求，双方密切合作明确系统总体设计方案，是整个设计过程的基础，结果影响后续阶段实施效率	编写需求说明书，包括数据流图、建立数据字典
概要设计	将需求分析得到的用户需求建立抽象的信息模型（概念模型），是现实世界的模型，便于与不熟悉计算机的用户交流，是数据库设计的关键	选择局部应用、逐一设计分 ER 图和 ER 图合并
逻辑设计	用具体 DBMS 实现用户需求，将概念结构转换为相应数据模型，根据用户处理要求、安全性考虑建立必要视图并优化数据模型	数据模型设计、E-R 图转换为关系模式、关系模式规范化、确定完整性约束、确定用户视图、反规范化设计

物理设计	为给定的逻辑数据模型选择高效、最适合的物理结构（主要指数据库的存储结构和存储方法），并对物理结构进行时间和空间效率方面的评价	
------	--	--

3.3. 概要设计（超级重点★★★★★）

概要设计常用策略是运用自顶向下方法进行需求分析，然后运用自底向上方法设计概念结构。下图体现了自底向上方法设计概念结构方法，可以看到首先设计出局部 E-R 模型，然后将各个局部 E-R 图合并起来形成全局 E-R 模型，最后将全局 E-R 模型进行优化，评审后得到最终的 E-R 模型，即概念模型。



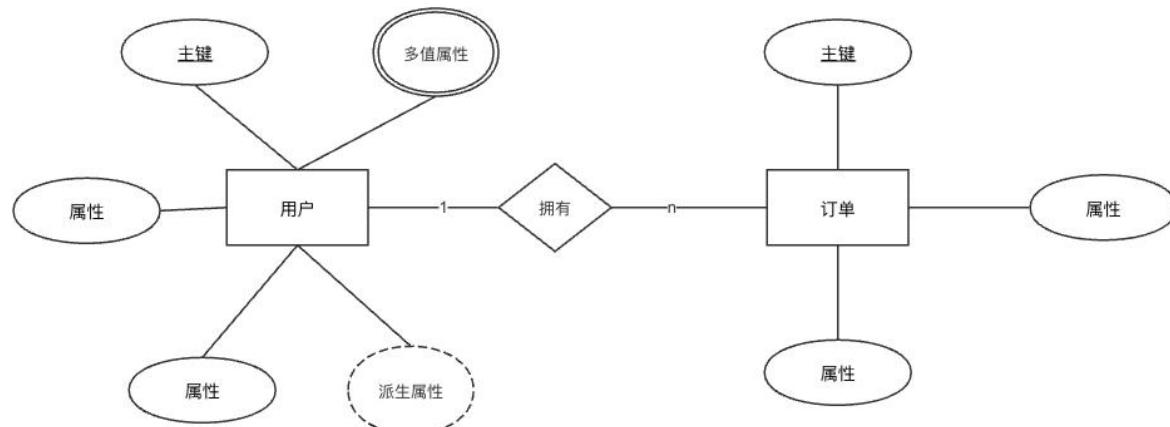
E-R 图的设计要依照上述的抽象机制，对需求分析阶段所得到的数据进行分类、聚合和概括，确定实体、属性和联系。概念结构设计工作步骤包括：选择局部应用、逐一设计分 E-R 图和 E-R 图合并。此章节属于选择案例重点考查的内容，特别是联系的类型，联系的转换必须熟练掌握，凯恩这里举了大量的例子辅助你理解。

E-R 模型也称为 E-R 图，它是描述概念世界，建立概念模型的实用工具。E-R 图三要素如下：

元素	表示方式	示例说明
实体（型）	用矩形框表示，框内标注实体名称	例如电商系统中的“用户”“商品”等实体，会分别用矩形框表示，框内写对应名称
属性	单值属性	假设“用户”实体有“姓名”属性，“姓名”就用椭圆形表示并与“用户”实体矩形框相连
	多值属性	如学员信息数据库中，“学员”实体的“个人兴趣”属性，可能有运动、电影等多个值，

		“个人兴趣”用双实线椭圆表示并与“学员”实体相连
派生属性	用虚线椭圆表示，从基本属性计算得出	像学员的“总成绩”“平均成绩”，从基本成绩属性计算得出，用虚线椭圆表示并与“学员”实体相连
实体之间的联系	用菱形框表示，框内标注联系名称，用连线将菱形框与有关实体相连，并在连线上注明联系类型(1:1, 1:n 或 m:n)	电商系统中，“用户”与“商品”实体间可能存在“购买”联系，用菱形框表示“购买”，分别与“用户”“商品”实体矩形框相连，连线上注明如“m:n”的联系类型

下图就是某电商系统的一个 E-R 图。



单单掌握上面的考点还不够，下面这些尚未考察的点，同样值得关注。

概念	定义	示例
弱实体	依赖于某个实体而存在的实体，依赖的对象为强实体	订单依赖于用户实体，订单是弱实体，用户实体是强实体
弱关系	一般与弱实体一起使用，仅弱实体才会涉及的关系	/
不完全概括	父实体的实例可以是子实体的实例，也可能不是任何子实体的实例，即父实体部分实例不属于任何子类别	职业作为父实体，其子实体为工程师、老师，存在部分职业不属于工程师和老师类别
全部概括	父实体的所有实例必须是某个子实体的实例，不存在独立的父实体实例，所有父实体实例都能归类到子实体中	人作为父实体，子实体为男人、女人，所有人必定属于男人或女人类别

3.3.1. 整体 E-R 图设计（重点★★★★★）

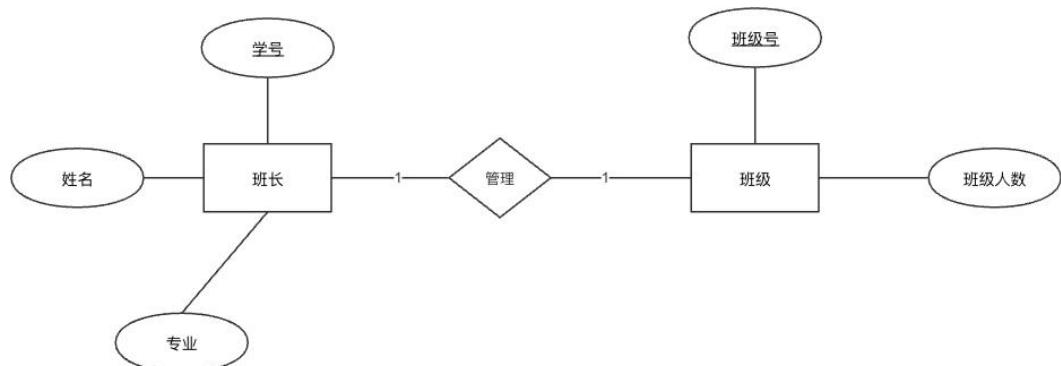
设计各子系统的局部 E-R 图设计理论过程是（1）确定局部视图的范围；（2）识别实体及其标识，确定实体之间的联系；（3）分配实体及联系的属性。各子系统的局部 E-R 图设计好后，下一步就是要将所有的分 E-R 图综合成一个系统的总体 E-R 图。这里仍然只会出选择题，真题掌握即可（记忆点——合并、消除冲突（属性/命名/结构）、消除冗余）。

步骤	内容		
合并	确定各局部 E-R 图的公共实体类型，以其为单位合并，直至所有相同实体类型合并完毕，得到全局 E-R 图		
消除冲突	因各子系统应用场景及设计人员不同，局部 E-R 图存在不一致问题即冲突，需消除冲突形成统一概念模型	属性冲突	包括属性域冲突和取值冲突，需各部门协商解决
		命名冲突	包含同名异义和异名同义，通过讨论、协商等行政手段解决
		结构冲突	同一对象在不同应用中抽象不同；同一实体在不同局部 E-R 图中属性个数或排列次序不同；实体间联系在局部 E-R 图中联系类型不同
消除冗余	冗余分为冗余属性（可由基本数据导出的数据）和冗余联系（可由其他联系导出的联系）		

3.4. 逻辑设计（超级重点★★★★★）

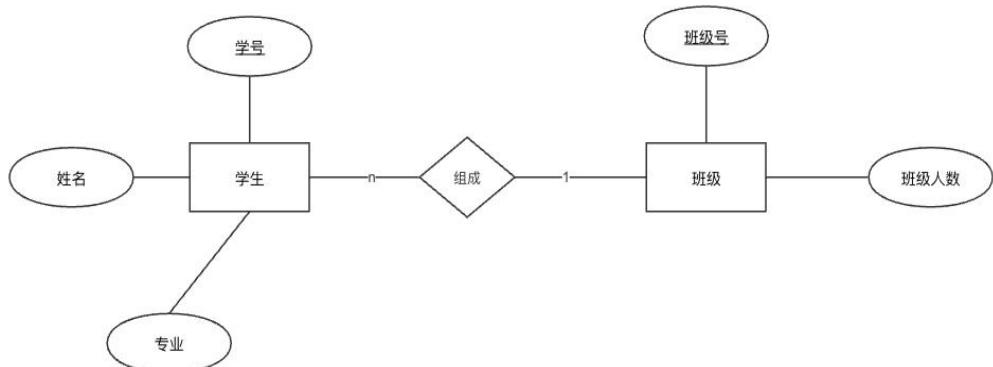
逻辑结构设计的任务就是把概念结构设计阶段完成的整体 E-R 模型转换为与选用的 DBMS 产品所支持的数据模型相符合的逻辑结构。进行数据库的逻辑设计，要将 E-R 模型向关系模型转换（可以理解为转换成表的列的设计，下面我就直接说表结构转换方便理解），范式转换也发生在这里。系分可能出案例题，架构只会出选择题。这里直接看比较抽象，先去看一下我的 B 站视频（搜索芝士架构凯恩）。

3.4.1.E-R 图转换关系模式（重点★★★★★）



这样一个 1:1 联系可以通过两种方法转换成符合范式的表结构。

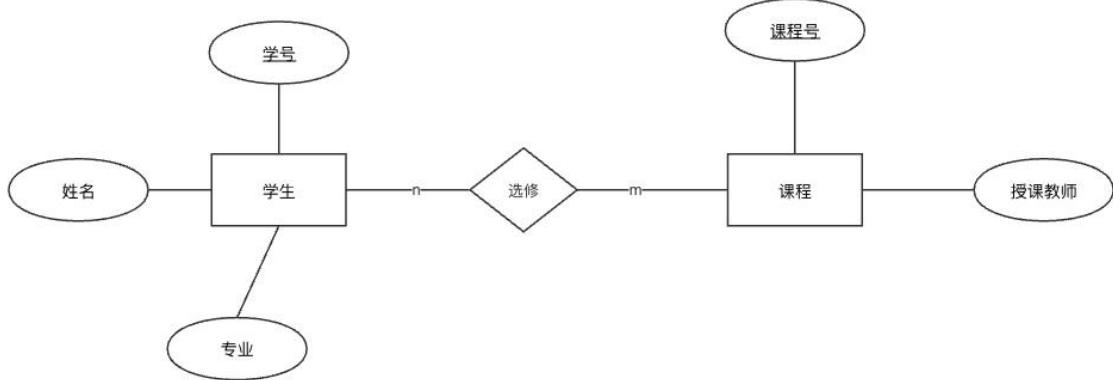
名称	转换方式	示例说明
转换为独立关系模式	关系的属性由联系相连各实体的主码以及联系本身的属性组成，主码是每个实体的主码	班长（学号，姓名，专业） 班级（班级号，班级人数） 管理（学号，班级号）
与联系一端实体的关系模式合并	将另一个实体的主码和联系本身的属性加入到该端实体关系模式的属性中	班长（学号，姓名，专业，班级号） 班级（班级号，班级人数）



这样一个 1:n 联系可以通过两种方法转换成符合范式的表结构。

名称	转换方式	示例说明
转换为独立关系模式	关系的属性由与该联系相连的各实体的主码以及联系本身的属性转换而来，主码是 n 端实体的主码	学生（学号，姓名，专业） 班级（班级号，班级人数） 组成（学号，班级号）

与 n 端实体对应关系模式合并	将联系本身的属性和 1 端实体的主码加入 n 端对应关系模式中	学生 (学号, 姓名, 专业, 班级号) 班级 (班级号, 班级人数)
-----------------	---------------------------------	--



一个 m : n 联系可以通过一种方法转换成符合范式的表结构。关系的属性由与该联系相连的各实体的主码以及联系本身的属性转换而来，各实体主码的组合是该关系的主码或关系主码的一部分。其实说白话就是把关系独立出来单独成表。

学生 (学号, 姓名, 专业)

课程 (班级号, 授课教师)

选修 (学号, 课程号)

3.4.2. 函数依赖 (重点★★★★★)

不用记定义，直接看下面例子。

例如：设一个职工关系为（职工号，姓名，性别，年龄，职务）。

解析：职工号函数决定姓名，或姓名函数依赖于职工号，记作“职工号→姓名”，职工号为该函数依赖的决定因素。

函数依赖还可以分为多种类型，见下表。其中部分函数依赖和传递函数依赖特别关键，后面讲到范式的时候会提到。

函数依赖类型	定义	例子
平凡函数依赖	若 $X \rightarrow Y$, 但 $Y \subseteq X$, 则称 $X \rightarrow Y$ 是平凡函数依赖	在职工关系 (职工号, 姓名, 性别, 年龄, 职务) 中, (职工号, 性别) \rightarrow 职工号, 因为职工号是 (职工号, 性别) 的子集, 所以是平凡函数依赖; (职工号, 性别) \rightarrow 性别同理
非平凡函数依赖	若 $X \rightarrow Y$, 且 $Y \not\subseteq X$, 则称 $X \rightarrow Y$ 是非平凡函数依赖	在职工关系 (职工号, 姓名, 性别, 年龄, 职务) 中, (职工号, 姓名) \rightarrow 性别, 性别不是 (职工号, 姓名) 的子集, 所以是非平凡函数依赖; (职工号, 姓名) \rightarrow (年龄, 职务) 同理
完全函数依赖	在关系模式 $R(U)$ 中, 如果 $X \rightarrow Y$, 并且对于 X 的任何一个真子集 X' , 都有 $X' \nrightarrow Y$, 则称 Y 对 X 完全函数依赖	在教师任课关系 (教工号, 姓名, 职称, 课程号, 课程名, 课时数, 课时费) 中, (职称, 课程号) 完全函数决定课时费, 即只有 (职称, 课程号) 能决定课时费, 单独的职称或课程号都不能决定课时费
部分函数依赖	在关系模式 $R(U)$ 中, 如果 $X \rightarrow Y$, 但 Y 不完全函数依赖于 X , 则称 Y 对 X 部分函数依赖	在教师任课关系 (教工号, 姓名, 职称, 课程号, 课程名, 课时数, 课时费) 中, (教工号, 课程号) 部分函数决定姓名, 因为教工号本身就可以决定姓名, 课程号在这里是多余的, 即存在教工号 \rightarrow 姓名, 所以 (教工号, 课程号) 是部分函数决定姓名; (教工号, 课程号) 部分函数决定课时数同理
传递函数依赖	在关系模式 $R(U)$ 中, 如果 $X \rightarrow Y$, $Y \rightarrow Z$, 且 $Y \nrightarrow X$, $Z \not\subseteq Y$, 则称 Z 对 X 传递函数依赖	在学生关系 (学号, 姓名, 性别, 系号, 系名, 系主任名) 中, 学号 \rightarrow 系号, 系号 \rightarrow 系名和系号 \rightarrow 系主任名, 且系号不能决定学号, 系名和系主任名不属于系号, 所以系名和系主任名传递依赖于学号

3.4.3. 无损联接分解 (次重点★★★☆☆)

这里特别爱考选择题, 所以凯恩标注了次重点。无损连接是指分解后的关系通过自然连接可以恢复成原来的关系, 即通过自然连接得到的关系与原来的关系相比, 既不多出信息、又不丢失信息。

无损联接分解的形式定义如下: 设 R 是一个关系模式, F 是 R 上的函数依赖集。 R 分解成数据库模式 $\delta = \{R_1, \dots, R_k\}$, 如果对 R 中每个满足 F 的关系 r 都有下式成立:

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \dots \bowtie \pi_{R_k}(r)$$

则称分解 S 相对于 F 是无损联接分解, 否则称为损失联接分解。这里的 \bowtie 表示自然连接。

这一段是废话不需要记忆, 可以理解直接看下面公式法

下面是一个很有用的无损联接分解判定定理，凯恩建议必须掌握：

设 $\rho = \{R_1, R_2\}$ 是 R 的一个分解， F 是 R 上的函数依赖集。那么分解 ρ 相对于 F 是无损级联分解的充要条件 $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ 或 $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ 。要注意的是，这两个条件只要有任意一个条件成立就可以了。具体做做真题或者看一下我的视频就能学会。

无损分解的公式法学到这里就可以了，有多个分解的情况更加复杂（表格法）不做展开，为了 1 分不值得，出到题目直接放弃。

3.4.4. 保持函数依赖（次重点★★★☆☆）

这里特别爱考选择题，所以凯恩标注了次重点，一般和无损连接一起考。所谓的保持函数依赖就是将分解之后的函数依赖的集合合并起来，只要合并之后，与原来的函数依赖集合是保持等价的，我们就会说是保持函数依赖的。无损和函数依赖没有关系，可能一个分解是无损的但是没有保持函数依赖，反之亦然。

设数据库模式 $\delta = \{R_1, \dots, R_K\}$ 是关系模式 R 的一个分解， F 是 R 上的函数依赖集， δ 中每个模式 R_i 上的函数依赖集是 F_i 。如果 $\{F_1, \dots, F_k\}$ 与 F 是等价的（即相互逻辑蕴涵），则称分解 δ 保持函数依赖。如果分解不能保持函数依赖，则 δ 的实例上的值就可能有违反函数依赖的现象。这里出过选择题，给你分解前模式和分解后的模式让你判断是否逻辑蕴含。这个通过作图和推理法都可以做，一般不会很难，做真题的时候看解析再学。

这里牵扯到一个逻辑蕴含的概念。设 F 是关系 $R(U)$ 中的一个函数依赖集合， X, Y 是 R 的属性子集，如果能从 F 这个函数依赖集合中推导出 $X \rightarrow Y$ ，则称 F 逻辑蕴含 $X \rightarrow Y$ ，或者说 $X \rightarrow Y$ 是 F 的逻辑蕴含。记作 $F \mid= XY$ 。 F 是集合一定要清楚。集合意味着里面有 $X \rightarrow Y, X \rightarrow Z$ 的等等依赖关系。

3.4.5. 规范化意义 (重点★★★★★)

此章节极大概率在案例中进行考察，凯恩建议熟记这里的不规范化导致的 4 大异常情况（冗删改查）以及例子。一旦在案例中出现至少 8 分！

设有一个关系模式

R (SNO, SNAME, CNAME, TNO, TNAME, TADDRESS)

其属性分别表示学生编号、学生姓名、课程编号、课程名、任课教师姓名和任课教师地址。

仔细分析一下，就会发现这个模式存在下列异常，这些异常需要掌握，可能在案例中进行考察，系分同学要格外关注：

问题类型	描述	示例
数据冗余	相同数据在关系中多次重复出现	某课程有 100 个学生选修，课程的任课教师姓名和地址在关系 R 中重复出现 100 次
修改异常	因数据冗余，修改数据时需多处修改，否则会导致数据不一致	修改教师地址时，需修改 100 个元组中的地址值，不然会出现地址值不一致
插入异常	因部分信息缺失，导致相关数据无法正常插入数据库	不知道听课学生名单时，教师的任课情况和家庭地址无法进入数据库，或需在学生姓名处插入空值
删除异常	删除某类数据时，会意外删除其他不应删除的数据	删除某门课程的任课教师信息时，学生信息也被删除

记忆口诀：绒绣茶山。毛绒绣出来的茶山玩具。绒（冗余）绣（修改）茶（插入）山（删除）。

3.4.6. 键/码/属性 (重点★★★★★)

下面的键/码/属性是讨论范式的基础，特别是主属性和非主属性。这个在 BCNF 范式的讨论中会再次强调。

概念	定义

主键 / 码	被数据库设计者选中，用于在同一实体集中区分不同实体的候选码，应选从不或极少变化的属性。一个实体集仅有一个主码
超码	一个或多个属性的集合，能在实体集中唯一标识一个实体，可能含多余属性。可唯一标识实体，但可能存在冗余属性
候选码	若超码的任一真子集不能成为超码，则为候选码，不包含多余属性
主属性	包含在任一候选码中的属性
非主属性	不包含在任一候选码中的属性

这里引入一种求候选键的快捷方法（这是必须掌握，选择题遇到就是送分），即图示法。

使用图示法求候选键，主要有两个步骤：

- (1) 将关系模式的函数依赖关系，用有向图的方式表示，其中顶点表示属性，弧表示属性之间的依赖关系。
- (2) 找出入度为 0 的属性集，并以该属性集为起点，尝试遍历有向图，若能正常遍历图中所有结点，则该属性集即为关系模式的候选键；若入度为 0 的属性集不能遍历图中所有结点，则需要尝试性地将一些中间顶点（既有入度，也有出度的顶点）并到入度为 0 的属性集中（真题中出现了这种情况），直至该集合能遍历所有顶点，则该集合为候选键。

下面这道题必会，给定关系 $R(A1, A2, A3, A4)$ 上的函数依赖集

$F = \{A1 \rightarrow A2, A3 \rightarrow A2, A2 \rightarrow A3, A2 \rightarrow A4\}$ ，现在要求 R 的候选键。需要针对函数依赖集画出有向图，如图 5-2 所示。这里的难点是可能有的题目会出现 $A2, A1 \rightarrow A4$ 的函数依赖，作图的时候你可以用特殊标记把他们标记出来（比如虚线，或者线段加粗共享一个箭头，防止搞错）。中找出入度为 0 的顶点，即 $A1$ 。通过尝试，可以发现从出发可以遍历所有顶点，因此， R 的候选键为 $A1$ 。

3.4.7.五大范式（超级重点★★★★★）

这块内容系分考生假如不明白不用去考了，案例必挂。

1NF、2NF、3NF、BCNF、4NF 全部在考试中出现过，选择案例都会考到，凯恩建议你一定要引起足够的重视，要学会判别和语言描述。（记忆：一范式，无重复，二范式，主键独，三范式，无传递，BCNF 传递无，四范式，多值除）。在判断一个关系属于哪种范式之前，一定要先找出它的主键！然后再进行分析。这里要严格注意，24 年系分案例已经出现范式判断和 BCNF 规范化内容！死记硬背也要记住 1NF、2NF、3NF、BCNF 的内涵和规范化方法，凯恩这里每个范式都举了例子，你一定要学会，并默写出来。

目前共定义了多个范式，分别为 1NF、2NF、3NF、BCNF、4NF 和 5NF。（23 年架构考试已经出现 BCNF、4NF），总结如下表所示（暂时看不懂不要慌不要急，下面有详细的讲解）。

范式名称	定义	判断关键	示例及规范化方法
第一范式（1NF）	所有属性只包含原子值，每个分量不可再分	属性是否为原子属性	教师职称情况关系中“高级职称人数”非原子属性，拆分为“教授”和“副教授”属性可满足 1NF
第二范式（2NF）	满足 1NF，且不存在非主属性对候选码的部分函数依赖；若每一个候选码都是单码则也满足	是否存在非主属性对候选码的部分函数依赖	学生关系（学号，姓名，性别，所在专业，课程号，课程名，成绩）中姓名等部分函数依赖于（学号，课程号），分解为学生（学号，姓名，性别，所在专业）、课程（课程号，课程名）、选课（学号，课程号，成绩）满足 2NF
第三范式（3NF）	满足 1NF，不存在非主属性对候选码的传递函数依赖；非主属性既不依赖也不传递依赖于任何候选码；不存在非主属性的关系模式一定为 3NF	是否存在非主属性对候选码的传递函数依赖	学生关系（学号，姓名，年龄，学校编号，学校地址，学校电话）存在传递函数依赖（学号->学校编号,学校编号->学校地址,学校电话），可分为学生（学号，姓名，年龄，学校编号）、学校（学校编号，地址，电话）或学生（学号，姓名，年龄）、学校（学校编号，地址，电话）、管理（学号，学校编号）满足 3NF
BCNF	满足 1NF，不存在任何属性对候选码的传递函数依赖，关系模式中任何函数依赖的左侧必须是码	函数依赖左侧是否为码	授课（教工号，学号，课程号）中存在教工号→课程号且教工号不是码，不是 BCNF 范式，分解为（学号，教工号）和（教工号，课程号）满足 BCNF

第四范式 (4NF)	是 BCNF 的推广, 针对有多值依赖的关系模型, 需将一个表中多个多值依赖拆分开	是否存在多个多值依赖	职工表 (职工编号, 职工孩子姓名, 职工选修课程) 存在两个多值依赖, 分为职工表 1 (职工编号, 职工孩子姓名) 和职工表 2 (职工编号, 职工选修课程) 满足 4NF
------------	---	------------	--

(1) 第一范式 (1NF)。在关系模式中, 当且仅当所有属性只包含原子值, 即每个分量都是不可再分的数据项, 则称满足 1NF。例如, 下表所示的教师职称情况关系就不满足 1NF。原因在于, 该关系模式中的“高级职称人数”不是一个原子属性。将其拆分为“教授”和“副教授”两个属性, 则就满足 1NF。

系名称	高级职称人数	
	教授	副教授
计算机系	6	10
电子系	3	5

(2) 第二范式 (2NF)。设一个关系为 R(U), 满足第一范式, 若 R 中不存在非主属性对候选码的部分函数依赖, 则称该关系是符合第二范式的, 即 $R \in 2NF$ 。推论: 若关系模式 $R \in 1NF$, 且它的每一个候选码都是单码, 则 $R \in 2NF$ 。

看下面例子感受一下 2NF 的规范化方法。

假定存在学生关系 (学号, 姓名, 性别, 所在专业, 课程号, 课程名, 成绩), 使其符合 2NF。

姓名, 性别, 所在专业部分函数依赖于 (学号, 课程号) 中的学号, 因此符合第一范式但不符合第二范式, 需分解为三个关系。

由于这个学生实体和课程实体是多对多关系, 根据前面多对多关系 ER 图到关系模式转换办法我们知道, 就是把联系单独提出来形成一个关系。

学生 (学号, 姓名, 性别, 所在专业)。

课程 (课程号, 课程名)。

选课（学号，课程号，成绩）。

经过分解，三个关系可以连接后仍得到原关系，且为无损分解和无损连接。

(3) 第三范式 (3NF)。设一个关系为 $R(U)$ ，满足第一范式，若 R 中不存在非主属性对候选码的传递函数依赖，则称该关系是符合第三范式的，即 $R \in 3NF$ 。

推论 1：如果关系模式 $R \in 1NF$ ，且它的每一个非主属性既不部分依赖，也不传递依赖于任何候选码，则 $R \in 3NF$ 。

推论 2：不存在非主属性的关系模式一定为 $3NF$ （所有属性都是主属性）。

有人说凯恩你写错了！第三范式是基于第二范式的，你怎么写了满足第一范式？其实不的，这是个推论，需要推导，推导过程很简单，通过反证法证明。

假设一个关系不满足 $2NF$ ，形式化表达如下： (A,B,C,D) ，其中 $(A,B) \rightarrow (C,D)$ 。假设存在 $B \rightarrow C$ （非主属性对码的部分依赖），因为 $(A,B) \rightarrow B$ （平凡依赖）， $(A,B) \rightarrow B$ ， $B \rightarrow C$ ，意味着存在 C 对码 (A,B) 的传递依赖。所以非 $2NF$ 一定是非 $3NF$ 。那么这个结论的等价结果（逆否命题）就是 $3NF$ 一定是 $2NF$ 。

看下面例子感受一下 $3NF$ 的规范化方法。

假定学生关系（学号，姓名，年龄，学校编号，学校地址，学校电话），使其符合 $3NF$ 。

因为 $(\text{学号}) \rightarrow (\text{姓名}, \text{年龄}, \text{学校编号}, \text{学院地点}, \text{学院电话})$ 和 $(\text{学号}) \rightarrow (\text{学校编号}) \rightarrow (\text{学校地址}, \text{学校电话})$ 存在非主属性对候选码的传递函数依赖。

这里有两个联系分别是学生实体和学校，他们之间是一对多关系，一个学校对应多个学生，一个学生对应一个学校。实体根据前面一对多联系转换成关系模式的方法，我们有两种办法来做规范化。

要将学生关系表分为如下关系的两个表。

学生（学号，姓名，年龄，学校编号）。

学校（学校编号，地址，电话）。

也可以分为三个表，把关系单独拿出来成表。

学生（学号，姓名，年龄）。

学校（学校编号，地址，电话）。

管理（学号，学校编号）。

(1) BCNF。如果对于关系模式 R 中存在的任意一个非平凡函数依赖 $X \rightarrow A$ ，都满足 X 是 R 的一个超键，那么关系模式 R 就属于 BCNF。这个书本写得很抽象。所以直接看例子理解。

根据定义一个简单的判断方法是在 BCNF 范式中，关系模式中任何函数依赖的左侧必须是码（候选码），不能存在非码的属性。

假设有关系模式 R(A,B,C)，候选码为 AB 和 AC，函数依赖有 $AB \rightarrow C$ ，以及 $AC \rightarrow B$ ，在这两个函数依赖中，左边的和都是候选码 (AB, AC)，所以关系模式属于 BCNF 范式。

说明关系模式授课（教工号，学号，课程号）是否属于 BCNF 范式。

注意这里有前提的：每名教师只教授一门课程，每门课程由若干名教师教授；若某一学生选定某门课程，会被分配该课程的一名教师；同样地，某名学生选修了某名教师就确定了所选课程的名称。所以有下面的推论。

(教工号，学号)，(学号，课程号)是候选码

但是由于存在教工号 \rightarrow 课程号。

推论判断：因为教工号不是码（是主属性），所以不满足 BCNF 范式。

(如何规范化)可将授课分解为两个关系模式，即 (学号, 教工号) 和 (教工号, 课程号)，它们之中不存在任何属性对候选码的部分函数依赖和传递函数依赖，所以符合 BCNF 范式。

当然有的同学说改成 (学号, 课程号) 和 (教工号, 课程号) 也可以。

(5) 第四范式 (4NF)。第四范式是 BCNF 的推广，是针对有多值依赖的关系模型所定义的规范化形式。把一个表中多个多值依赖拆分开来。这里做不形式化的表达，单单讲不满足 4NF 的例子，以及如何分解，举个例子好理解一点。

职工表 (职工编号, 职工孩子姓名, 职工选修课程)

在这个表中，同一个职工可能会有多个职工孩子姓名。同样，同一个职工也可能会有多个职工选修课程。由于存在两个多值依赖，就会导致表中数据冗余或者插入异常等其他问题。即这里存在着多值事实，不符合第四范式。

如果要符合第四范式，只需要将上表分为两个表，使它们只有一个多值事实，例如职工表 1 (职工编号, 职工孩子姓名)，职工表 2 (职工编号, 职工选修课程)，两个表都只有一个多值事实，所以符合第四范式。

3.4.8. 典型例题 (超级重点★★★★★)

(凯恩模拟) 某高校存在关系模式 教师授课(教师号, 姓名, 职称, 课程号, 课程名, 学分, 教科书名)。已知属性均为原子值 (无重复组)。已收集到的业务规则与函数依赖如下：
 $F = \{ \text{教师号} \rightarrow \text{姓名}, \text{教师号} \rightarrow \text{职称}, \text{课程号} \rightarrow \text{课程名}, \text{课程号} \rightarrow \text{学分}, \text{课程号} \rightarrow \text{教科书名} \}$ 。
假设一名教师可讲授多门课程，一门课程亦可能由多名教师讲授 (不同班次)。

问：指出这个关系模式的主码。这个关系模式是第几范式，并给出依据？将其分解为满足 3NF 要求的关系模式。

1. 关于候选码与属性分类：由 F 可见，教师号唯一决定教师侧信息 (姓名、职称)，课程号唯一决定课程侧信息 (课程名、学分、教科书名)。要在“教师授课”这一事实表中唯一标识一条记录，既要区分教师也要区分课程，因此将 {教师号, 课程号} 组合起来才能区分“同一教师教授不同课程”以及“同一课程由不同教师教授”的情形。检验： $\{\text{教师号}, \text{课程号}\}^+ =$

全属性（由教师号推出姓名、职称；由课程号推出课程名、学分、教科书名；再联合自身两码即覆盖全部），且任何真子集（仅教师号或仅课程号）闭包都不能覆盖对方一侧信息，因此是最小超码，即候选码。于是主属性为两码本身，其余均为非主属性。

2. 题干已说明各属性为原子值，因而满足 1NF。检视对候选码 $K=\{\text{教师号}, \text{课程号}\}$ 的依赖：存在 $\text{教师号} \rightarrow (\text{姓名}, \text{职称})$ 与 $\text{课程号} \rightarrow (\text{课程名}, \text{学分}, \text{教科书名})$ ，这些非主属性并非依赖于整个 K ，而是依赖于其真子集（教师号或课程号）。这正是部分函数依赖的典型情形，违背 2NF “每个非主属性必须完全依赖于候选码”的要求。不存在非主属性经由其他非主属性传递自码（如课程名再决定别的属性）的事实依赖，因此是否满足 3NF 已无从谈起；综上，当前最高仅为 1NF。

3. 将以“教师号”为决定属性的依赖归入教师实体表，形成 $\text{教师}(\text{教师号}, \text{姓名}, \text{职称})$ ；将以“课程号”为决定属性的依赖归入课程实体表，形成 $\text{课程}(\text{课程号}, \text{课程名}, \text{学分}, \text{教科书名})$ ；剩余用于表示多对多关联的码对，形成联系表 $\text{教师授课}(\text{教师号}, \text{课程号})$ 。三个子模式中，教师号与课程号分别作为各自实体表主码，联系表主码为复合码 $\{\text{教师号}, \text{课程号}\}$ 。该分解满足 3NF/BCNF：各表中每个非主属性都完全且仅依赖于其所在表的键，不存在部分或传递依赖。

答：主码为：（教师号、课程号）。这个关系模式属于第 1 范式，因为存在部分依赖。

分解方法为：教师表（教师号，姓名，职称）课程表（课程号，课程名，学分，教科书名）
教师授课表（教师号，课程号）

（凯恩模拟）某高校数据库中有学生表：

学生(学号, 姓名, 年龄, 性别, 系名, 专业名, 班号)。

其中约束条件为：

一个系可以包含多个专业；

每个专业下可以有多个班；

班号在全校范围内不重复；

每个学生只属于一个班级。

在业务使用中，系统经常需要根据系名和班号进行查询，如“查询某系某班所有学生名单”。

为提高查询效率，考虑在列 系名 与 班号 上建立一个非聚合索引。有两种选择：

方法 1：建立复合索引，列顺序为(系名，班号)。

方法 2：建立复合索引，列顺序为(班号，系名)。

问：请判断哪种方法更合理，并简要说明理由。

在数据库索引设计中，复合索引的使用遵循一个非常重要的规则，即最左前缀匹配原则。

所谓最左前缀，是指复合索引底层的排序和存储是按照定义时的列顺序依次进行的。例如建立(系名，班号)索引时，索引首先按照“系名”进行排序和组织，同一“系名”下再根据“班号”排序。

因此，在查询时，数据库优化器能够利用索引的范围扫描或精确匹配能力，前提是查询条件必须从最左列开始连续匹配。换句话说，只有当“系名”出现在查询条件里时，索引的优势才能发挥出来，之后再结合“班号”可以进一步缩小检索范围。这就是最左前缀原则的基本含义。

结合题意来分析，学生表中班号在全校范围内是唯一的，如果我们采用 (班号，系名) 作为索引顺序，虽然在 WHERE 班号=... 或 WHERE 班号=... AND 系名=... 这样的查询中能够使用索引，但由于班号本身已经唯一定位学生班级，所以附加的“系名”过滤条件没有实际作用。这种情况下 (班号，系名) 实际上退化成了一个单列索引 (班号)，系名的存在没有带来额外性能提升。而当查询条件只有 WHERE 系名=... 时，由于违反了最左前缀原则 (跳过了班号列)，该索引完全无法使用，只能退化为全表扫描或依赖其他索引。

相比之下，若我们选择 (系名，班号) 作为复合索引，就可以同时支持两类常见的查询：一是仅按系名进行的范围查询，即 WHERE 系名=...；二是按系名和班号组合的精确查询，即 WHERE 系名=... AND 班号=...。这完全契合了题干所强调的“经常需要按系名和班号查询”的

业务场景。数据库在执行时会先利用索引快速锁定某个系的学生记录区，再在该范围内通过班号条件进一步过滤，效率更高，逻辑也更符合实际业务操作。

方法 1 更合理，即索引列顺序为 (系名, 班号)。

3.5.数据库的控制功能（重点★★★★★）

这一章节常考概念，系分容易出案例题，架构一般不会。要想使数据库中的数据达到应用的要求，必须对其进行各种控制，这就是 DBMS 的控制功能，包括并发控制、性能优化、数据完整性和安全性，以及数据备份与恢复等问题。这些技术虽然给人们的感觉是边缘性技术，但对 DBMS 的应用而言，却是至关重要的。

3.5.1.并发控制（重点★★★★★）

在多用户共享系统中，许多事务可能同时对同一数据进行操作，称为并发操作。此时，DBMS 的并发控制子系统负责协调并发事务的执行，保证数据库的完整性不受破坏，同时，避免用户得到不正确的数据。

3.5.2.事务的基本概念（重点★★★★★）

DBMS 运行的基本工作单位是事务，事务是用户定义的一个数据库操作序列，这些操作序列要么全做，要么全不做，是一个不可分割的工作单位。事务具有以下特性：（ACID 目前案例还没考过，这里的原理需要记住，可能出概念题，至少 8 分）

特性名称	含义	示例
原子性 (Atomicity)	事务是数据库的逻辑工作单位，事务包含的一组更新操作是原子不可分的，是一个整体，不能部分地完成	强调事务中的操作要么全部执行成功，要么全部不执行

一致性 (Consistency)	使数据库从一个一致性状态变到另一个一致性状态	转账操作中，各账户金额必须平衡。与原子性密切相关，由事务的隔离性来表示，逻辑上不独立
隔离性 (Isolation)	一个事务的执行不能被其他事务干扰，一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰，即使多个事务并发执行，看上去也像每个事务按串行调度执行一样，也称为可串行性	强调并发事务之间的相互隔离，不互相干扰
持久性 (Durability)	也称为永久性，事务一旦提交，改变就是永久性的，无论发生何种故障，都不应该对其有任何影响	比如订单提交成功后，相关数据的改变会永久保存，不会因故障等原因丢失

3.5.3. 数据一致性问题 (重点★★★★★)

数据不一致问题以往系分案例考过填空题，有概率改成案例概念题，这里的例子和三个不一致的场景必须掌握。数据库的并发操作会带来一些数据不一致问题，例如，丢失修改、读“脏数据”和不可重复读等。实际上可以概括为“写-写”并发场景和“读-写”并发场景。

问题类型	描述	示例(结合下面的表格来看)				
丢失修改 (写覆盖)	事务 A 与事务 B 从数据库中读入同一数据并修改，事务 B 的提交结果破坏了事务 A 提交的结果，导致事务 A 的修改被丢失。这里又可以细分为第一类更新丢失问题：A 事务回滚时，把已经提交的 B 事务的更新数据覆盖了。第二类更新丢失问题：A 事务覆盖 B 事务已经提交的数据，造成 B 事务所做操作丢失。	<p>有 T1、T2 两个事务，执行顺序为：T1 读 A；T2 读 A；T1 执行 $A = A - 5$，写回；T2 执行 $A = A - 8$，写回。则“③$A = A - 5$，写回”操作会被“$A = A - 8$，写回”操作覆盖掉，“③$A = A - 5$，写回”将不起任何作用。</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">T1</td> <td style="text-align: center;">T2</td> </tr> <tr> <td> 读 A=10 ↓ A=A-5 ↓ 写回 </td> <td> 读 A=10 ↓ A=A-8 ↓ 写回 </td> </tr> </table>	T1	T2	读 A=10 ↓ A=A-5 ↓ 写回	读 A=10 ↓ A=A-8 ↓ 写回
T1	T2					
读 A=10 ↓ A=A-5 ↓ 写回	读 A=10 ↓ A=A-8 ↓ 写回					

读“脏数据” (读回滚)	<p>事务 A 修改某一数据，并将其写回磁盘，事务 B 读取同一数据后，事务 A 由于某种原因被撤销，这时事务 A 已修改过的数据恢复原值，事务 B 读到的数据就与数据库中的数据不一致，是不正确的数据，称为“脏数据”。</p>	<p>有 T1、T2 两个事务，执行顺序为：T1 读 A；T1 将 A 修改为新值并写回；T2 读 A = 新值；T1 撤销，A 恢复原值。则 T2 中“读 A = 新值”就是读的脏数据，比如假设初始 A 为 100，T1 改为 70，T2 读 A = 70，T1 撤销后 A 变回 100，T2 读的 70 就是脏数据。</p>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;">T1</th> <th style="text-align: center; padding: 5px;">T2</th> </tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;"> 读 A=20 ↓ A=A+50 ↓ 写回 ↓ 回滚 ↓ 读 A=20 </td><td style="text-align: center; padding: 5px;"> 读 A=70 </td></tr> </tbody> </table>	T1	T2	读 A=20 ↓ A=A+50 ↓ 写回 ↓ 回滚 ↓ 读 A=20	读 A=70
T1	T2						
读 A=20 ↓ A=A+50 ↓ 写回 ↓ 回滚 ↓ 读 A=20	读 A=70						
不可重复读 (读更新)	<p>事务 A 读取数据后，事务 B 执行了更新操作，事务 A 前后两次读取结果就发生变化，造成了数据不一致性。</p>	<p>有 T1、T2 两个事务，执行顺序为：T1 读 A 并进行运算；T2 对 A 进行更新操作并写回；T1 再次读 A 并进行运算，然后验算，发现两次运算结果不同。因为 T1 第一次读 A 后，T2 修改了 A 的值，T1 第二次读 A 时用的更新后的值，导致验算结果不正确。</p>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;">T1</th> <th style="text-align: center; padding: 5px;">T2</th> </tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;"> 读 A=20 ↓ 读 B=30 ↓ 求和 50 ↓ 读 A=70 ↓ 读 B=30 ↓ 求和 100 ↓ 验算不对 </td><td style="text-align: center; padding: 5px;"> 读 A=20 ↓ A=A+50 ↓ 写 A=70 </td></tr> </tbody> </table>	T1	T2	读 A=20 ↓ 读 B=30 ↓ 求和 50 ↓ 读 A=70 ↓ 读 B=30 ↓ 求和 100 ↓ 验算不对	读 A=20 ↓ A=A+50 ↓ 写 A=70
T1	T2						
读 A=20 ↓ 读 B=30 ↓ 求和 50 ↓ 读 A=70 ↓ 读 B=30 ↓ 求和 100 ↓ 验算不对	读 A=20 ↓ A=A+50 ↓ 写 A=70						
幻读 (读插入)	<p>T1 读取某个范围的数据，T2 在这个范围内插入新的数据，T1 再次读取这个范围的数据，此时读取的结果和第一次读取的结果不同。本质上也属于不可重复读的情况。只不过是插入导致的。</p>	<p>例如 T1 读取年龄在 20 - 30 岁之间的学生记录，T2 插入了一条年龄为 25 岁的学生记录，T1 再次读取年龄在 20 - 30 岁之间的学生记录时，结果比第一次多了一条记录。</p>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center; padding: 5px;">T1</th> <th style="text-align: center; padding: 5px;">T2</th> </tr> </thead> <tbody> <tr> <td style="text-align: center; padding: 5px;"> 读所有行 ↓ 计数 ↓ 读所有行 ↓ 计数 ↓ 验算不对 </td><td style="text-align: center; padding: 5px;"> 插入数据 </td></tr> </tbody> </table>	T1	T2	读所有行 ↓ 计数 ↓ 读所有行 ↓ 计数 ↓ 验算不对	插入数据
T1	T2						
读所有行 ↓ 计数 ↓ 读所有行 ↓ 计数 ↓ 验算不对	插入数据						

3.5.4. 封锁技术（重点★★★★★）

数据库事务锁很有可能在案例中出现，可能结合 MySQL 事务隔离机制出填空或者概念题。

处理并发控制的主要方法是采用封锁技术，主要有两种封锁，分别是 X 封锁和 S 封锁。

封锁类型	含义	特点	操作权限	解除封锁要求
排他型封锁 (X 封锁)	事务 T 对数据对象 A 实现 X 封锁	只允许一个事务独锁某个数据，具有排他性	事务 T 可读取和修改数据 A；其他事务在 T 解除 X 封锁前，不能对数据 A 进行任何类型的封锁	事务 T 完成对数据 A 的操作后解除
共享型封锁 (S 封锁)	事务 T 对数据 A 实现 S 封锁	允许并发读，但不允许修改	事务 T 只能读取数据 A，不能修改；在所有 S 封锁解除前，任何事务不能对数据 A 实现 X 封锁	所有对数据 A 加 S 封锁的事务完成读取操作后解除

虽然只有两种锁，但是结合不同的场景就会有不同的功效，在多个事务并发执行的系统中，主要采取封锁协议来进行处理，常见的封锁协议如下：这里大致了解就行，基本不会做深入考察。

封锁协议	具体内容	能解决的问题	不能解决的问题
一级封锁协议	事务 T 在修改数据 R 之前必须先对其加 X 锁，直到事务结束才释放	防止丢失修改，保证事务 T 可恢复	不能保证可重复读和不读“脏数据”，存在事务 T1 读数据 → 事务 T2 加 X 锁并改数据释放锁 → 事务 T1 再读数据导致的问题
二级封锁协议	一级封锁协议基础上，事务 T 在读取数据 R 之前先对其加 S 锁，读完后即可释放 S 锁	防止丢失修改、防止读“脏数据”	不能保证可重复读
三级封锁协议	一级封锁协议基础上，事务 T 在读取数据 R 之前先对其加 S 锁，直到事务结束才释放	防止丢失修改、读“脏数据”，能保证可重复读	无
两段锁协议	所有事务必须分两个阶段对数据项加锁和解锁，扩展阶段：对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；收缩阶段：释放一个封锁之后，事务不能再申请和获得任何其他封锁	若并发执行的所有事务均遵守，则任何并发调度策略都是可串行化的	遵守该协议的事务可能发生死锁

这里需要解释的是二级封锁协议。事务 T 在修改数据 R 之前必须先对其加 X 锁，直到事

务结束才释放。一级封锁协议可防止丢失修改，并保证事务 T 是可恢复的，但不能保证可重复读和不读“脏数据”。这里同学可能会有疑问，不是说 X 是独占的吗！加了锁不就串行化了吗，不会有可重复和脏数据啊。这里要注意，有这样一种情况，事务 T1 读数据 → 事务 T2 加 X 锁并改数据释放锁 → 事务 T1 再读数据。换句话说，T2 加 X 锁的时候不能够判断是否已经有事务在读了，这个情况会导致问题，这是一级封锁协议的关键问题。

显然，使用封锁技术来解决并发控制问题，存在一个封锁粒度问题。所谓封锁粒度，是指被封锁数据对象的大小，这里记住结论，封锁粒度小则并发性高，但开销大；封锁粒度大则并发性低但开销小，综合平衡照顾不同需求，以合理选取适当的封锁粒度是很重要的。

3.6.数据库性能优化（次重点★★★☆☆）

这里极有可能出现案例题，让你写出常见的数据库优化手段。

3.6.1. 反规范化（重点★★★★★）

从某种意义上来说，非规范化（反规范化）可以改善系统的性能。在进行数据库设计时，可以考虑合理增加冗余属性，以提升系统性能。书本知识点归纳如下。

类别	详情
反规范化提升性能的措施	将常用的计算属性（如总计、最大值等）存储到数据库实体中
	重新定义实体，以减少外部属性数据或行数据的开支
	将关系进行水平或垂直分割，以提升并行访问度
反规范化带来的问题	数据冗余增加，相同数据在多个地方重复存储
	更新异常，因数据冗余，更新一处数据可能遗漏其他重复存储处，导致数据不一致
	插入异常，为满足反规范化结构要求，可能需先插入不必要数据
	删除异常，删除某些数据可能意外删除其他有用信息

3.6.2. 索引优化 (重点★★★★★)

索引相关知识也是案例常考内容，也是书本上的内容，它的使用准则建议熟记，会在案例中让你默写概念。索引是提高数据库查询速度的利器，而数据库查询往往又是数据库系统中最频繁的操作，因此，索引的建立与选择对数据库性能优化具有重大意义。索引的建立与选择可遵循以下准则。

建议内容	具体描述
索引选用属性原则	选用经常作为查询，不常更新的属性建立索引；避免对常更新属性建立索引，因其严重影响性能
索引数量影响	关系上索引过多会影响 UPDATE、INSERT 和 DELETE 性能，因关系更新时所有索引都需相应调整
索引优化策略	分析每个重要查询使用频度，找出使用最多的索引并进行优化
小数据量关系处理	数据量非常小的关系不必建立索引，关系扫描更快且消耗系统资源更少

3.6.3. 完整性约束 (重点★★★★★)

数据库完整性由各种各样的完整性约束来保证，完整性约束可以通过 DBMS 或应用程序来实现，基于 DBMS 的完整性约束作为关系模式的一部分存入数据库中。此章节仍然是案例重点考察知识点，考察方式为概念默写。特别是下面表格的补充部分，需要理解记忆和掌握。

约束手段	描述	补充
实体完整性	实体完整性规则 (Entity Integrity Rule) 是指关系的主属性，即主码 (主键) 的组成不能为空，也就是关系的主属性不能是空值 (NULL)。	/
参照完整性	若基本关系中含有与另一基本关系 S 的主键 PK 相对应的属性组 FK (FK 称为 R 的外键)，则参照完整性要求，对及中的每个元组在 FK 上的值必须是 S 中某个元组的 PK 值，或者为空值。	插入删除问题 (级联/受限/置空/递归)

触发器	触发器是一个数据库对象，当指定数据操作语言操作发生时（触发事件），该对象可以自动执行一个或多个 SQL 语句（触发操作）。	可以在一个表上定义一个或多个触发器以便在 INSERT、UPDATE 或 DELETE 触发事件发生之后进行操作（不能直接对 SELECT 定义触发器，因为 SELECT 只是读取数据，不改变表的状态）。
用户定义完整性	针对特定数据库应用所定义的约束条件，由用户根据实际业务需求来制定。它可以对表中的列数据进行各种限制，如数据类型、取值范围、数据格式等，以确保数据满足业务规则。	例如，在员工信息表中，规定员工的年龄必须在 18 – 60 岁之间，或者规定某一列只能输入特定的值等。

3.6.4. 触发器示例（重点★★★★★）

系分案例考过触发器填空，所以必须重视起来。以 MySQL 为例：

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
[trigger_body]
```

```
CREATE TRIGGER before_insert_user
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    SET NEW.create_time = NOW();
END;
```

这里，在触发器中，可以通过虚拟表访问行数据（NEW,OLD），如下表所示。

触发器	虚拟表
INSERT 触发器	只能用 NEW.column_name（新插入的值）
UPDATE 触发器	可以用 OLD.column_name（更新前的值）和 NEW.column_name（更新后的值）
DELETE 触发器	只能用 OLD.column_name（被删除的值）

3.6.5. 典型例题（重点★★★★★）

触发器考过了，存储过程也能会考，下面答案划线部分要记忆。

(凯恩模拟) 设有四个关系

学生表 Student(sno,sname,sex,dept)

课程表 Course(cno,cname,precno,ceredit,tno)

教师表 Teacher(tno,tname,sex)

选课表 SC(sno,cno,grade)

问：编写一个触发器，当对课程表进行修改时其学分只能在 1 到 5 分之间。

```
CREATE TRIGGER check_course_credit_insert_update
BEFORE INSERT OR UPDATE ON Course
FOR EACH ROW
BEGIN
    IF NEW.ceredit < 1 OR NEW.ceredit > 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '课程学分必须在 1 到 5 之间';
    END IF;
END
```

(BEFORE INSERT OR UPDATE : 保证在写入或修改前校验。)

NEW.ceredit : 表示即将写入的新值。

SIGNAL SQLSTATE '45000' : 抛出一个自定义错误，阻止非法数据插入/更新。)

问：编写一个存储过程，依据输入的课程号参数，统计该门课程的平均分和总分，

要求使用游标，不可以使用 sum 和 avg 命令。

```
CREATE PROCEDURE stat_course_score(IN p_cno VARCHAR(10))
BEGIN
    -- 定义变量
    DECLARE v_grade INT;
    DECLARE v_sum INT DEFAULT 0;
    DECLARE v_count INT DEFAULT 0;
    DECLARE done INT DEFAULT 0;

    -- 定义游标
```

```
DECLARE cur CURSOR FOR
    SELECT grade
        FROM SC
    WHERE cno = p_cno;
```

-- 处理游标结束的条件

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

-- 打开游标

```
OPEN cur;
```

-- 循环取数

```
read_loop: LOOP
    FETCH cur INTO v_grade;
    IF done = 1 THEN
        LEAVE read_loop;
    END IF;
```

-- 累加成绩

```
    SET v_sum = v_sum + v_grade;
    SET v_count = v_count + 1;
END LOOP;
```

-- 关闭游标

```
CLOSE cur;
```

-- 输出结果

```
SELECT v_sum AS 总分,
```

```
    CASE WHEN v_count > 0 THEN v_sum / v_count ELSE NULL END AS 平均分;
```

```
END
```

(done 初始值是 0，表示“游标还没读完”。CONTINUE HANDLER FOR NOT FOUND 是一种错误处理机制：当游标 FETCH 失败（即结果集没有更多行了），就会触发这个处理，把 done 设置成 1。在循环里，我们每次 FETCH 一行，如果发现 done=1，就说明读完了，

于是 LEAVE read_loop; 退出循环。)

3.7. 备份与恢复技术（次重点★★★★☆）

这一章节不太重要很少来考，看过算过，可以用自己的语言描述出来即可。数据库备份有多种分类方式，如下表所示。

分类方法	子分类	解读
按备份的实现方式	物理备份	指直接拷贝数据库的数据文件、日志文件等物理文件来进行备份。比如在关系型数据库中，将存储数据的 .mdf (SQL Server 数据文件)、.dbf (部分数据库表文件) 等文件以及日志文件复制到其他存储位置。这种备份方式与数据库的物理存储结构紧密相关，恢复时也是通过这些物理文件来恢复数据库到特定状态，恢复速度相对较快，常用于需要快速恢复数据库的场景。
	逻辑备份	通过数据库的导出工具，将数据库中的数据以逻辑对象（如表、视图、存储过程等）的形式导出为特定格式的文件，如 SQL 脚本文件等。它不直接操作数据库的物理文件，而是按照数据库的逻辑结构来备份数据。恢复时需执行这些逻辑文件中的指令来重建数据和对象，相对物理备份恢复速度可能较慢，但灵活性较高，可跨平台和数据库版本使用，常用于数据迁移、数据子集备份等场景。
按备份数据量情况	完全备份	对整个数据库进行完整的备份，包括所有的数据表、索引、视图、存储过程等数据库对象以及数据。它是最基础的备份方式，恢复时可以直接从完全备份中恢复整个数据库到备份时的状态，但备份所需时间长、占用存储空间大，因为每次都要备份全部数据。
	增量备份	只备份自上次备份（可以是完全备份或上一次增量备份）以来发生变化的数据。相比完全备份，它备份的数据量较小，备份速度快，占用空间少。但恢复时需要依次使用完全备份以及后续的所有增量备份来恢复数据，恢复过程相对复杂且耗时，如果其中某个增量备份损坏，可能影响恢复。
	差异备份	备份自上次完全备份以来发生变化的数据。它介于完全备份和增量备份之间，备份的数据量比增量备份大，但比完全备份小，备份速度也相对较快。恢复时，只需使用完全备份和最近一次的差异备份即可恢复数据，恢复过程比增量备份简单，在数据恢复和备份效率上取得一定平衡。

4.上篇-Web 与移动应用系统分析与设计(新教材★ ★★★★★)

这一章在书上分为两节来讲，同时内容非常杂非常乱，从协议讲到框架，讲到前端技术后端技术甚至讲到测试，但是能提炼出的案例的考点实属不多。这里应用开发框架、服务端技术，客户端技术、测试这三节很水，直接略过。

4.1.Web 设计原则（超级重点★★★★★）

之前在讲面向对象设计的时候，凯恩提到过面向对象的设计原则也就是 SOLID 原则（忘了的再往前翻一翻）。这里教材提到的 Web 设计原则实际上和 SOLID 非常接近。这一章节的内容非常符合案例的出题思路，比较容易给你指定场景，问你满足了或者违反了哪些设计原则。

原则名称	定义	核心思想
分离关注点	根据功能类型拆分软件逻辑	业务逻辑与基础设施 / UI 分离，降低耦合
封装	隔离组件内部实现，通过接口交互	限制外部直接访问状态，维护公共协定
依赖反转	依赖抽象而非实现，编译时依赖方向与运行时相反	接口解耦，支持替换实现
显式依赖	通过构造函数明确声明依赖	确保客户端明确依赖项，避免隐藏需求
单一责任	一个组件/类只负责单一功能	降低变更风险，提高可维护性
避免自我重复	同一逻辑仅实现一次	减少维护成本，避免不一致
持久性无知	业务模型不依赖具体持久化技术	支持多存储方式，灵活扩展
有界上下文	独立概念模块，通过接口通信	解耦大型系统支持独立演进

这里重点关注依赖反转/倒置，做过 Java 开发的会比较熟悉。这里一句话总结就是，模块

依赖角度来看，是高层模块不应该依赖低层模块，二者都应该依赖其抽象。类 A 和类 B 同时依赖抽象接口 B。更深入的来说是，这种做法，让类 A 可以在运行时调用类 B，而类 B 又在编译时依赖于类 A 控制的接口。运行时，程序执行的流程保持不变，但接口引入意味着可以轻松对这些接口进行替换。

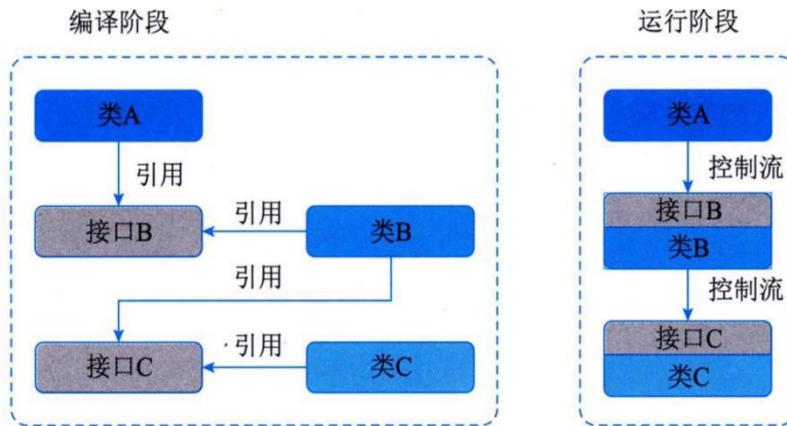


图 16-2 反转依赖关系图

4.2. Web 架构技术选型（超级重点★★★★★）

这里有两个核心考点，一个是三种架构（整体/微服务/无服务架构）的基本概念和优缺点，一种是它们两两对比。

架构类型	基本概念	优点	缺点	适用场景
整体 / 单体	单一代码库，所有组件耦合部署	开发 / 测试 / 部署简单；适合轻量级应用	扩展性差；维护成本高；单点故障风险	简单内部工具、低流量公共服务
微服务	拆分为独立服务，通过 API 通信；独立数据库	灵活扩展；支持持续交付；松耦合	管理复杂度高；服务间通信开销；事务一致性挑战	复杂高流量应用（如社交网络、电商平台）
无服务器	底层基础设施由云管理，代码按需执行	按需付费；自动扩展；减少运维成本	供应商锁定；冷启动延迟；复杂逻辑难以管理	多媒体处理、实时流、物联网消息、CI/CD 管道

系分实际上非常喜欢考给你一个具体的场景，问你选择哪种技术路线或架构路线，此时下面的这个表就可以派上用场。这里可以从部署、可扩展性（资源角度），可维护性，可靠性等角度来说。

对比维度	整体式架构	微服务架构	无服务器架构
部署方式	单一程序包部署	独立服务容器化部署	代码片段动态执行
扩展性	垂直扩展（增加服务器资源）	水平扩展（增加服务实例）	自动弹性扩展
维护成本	高（牵一发而动全身）	中（独立服务更新）	低（无服务器管理）

4.3.其他架构模式（次重点★★★☆☆）

这里的架构模式和前一节的架构模式实际上有交叉的（不知道书本这样写的意欲为何），这里单独列出作为补充。

4.3.1. 和视图相关的架构（次重点★★★☆☆）

MVC 和 MVP 还有 MVVM 的差别容易考的点，这里要从数据流和数据交互的角度来谈，如下表所示。

维度	MVC	MVP	MVVM
核心组件	模型（Model）、视图（View）、控制器（Controller）	模型（Model）、视图（View）、呈现器（Presenter）	模型（Model）、视图（View）、视图模型（ViewModel）
概念	模型：数据存储与业务逻辑 视图：用户界面展示 控制器：处理用户输入，协调模型与视图	模型：数据存储与业务逻辑 视图：用户界面交互接口 呈现器：处理逻辑，控制视图更新	模型：数据存储与业务逻辑 视图：绑定数据的 UI 视图模型：数据转换与双向绑定

数据流	单向：视图 → 控制器 → 模型 → 视图更新（通过控制器）	双向：视图 ↔ 呈现器 ↔ 模型（呈现器主动更新视图）	双向：视图 ↔ 视图模型 ↔ 模型（自动同步，基于绑定）
数据交互	视图可直接调用控制器，可能访问模型	视图通过接口与呈现器通信，不直接访问模型	视图通过数据绑定与视图模型交互，无直接逻辑调用
图例	<pre> graph LR View[View] --> Controller[Controller] Controller --> Model[Model] </pre>	<pre> graph LR View[View] --> Presenter[Presenter] Presenter <--> Model[Model] </pre>	<pre> graph LR View[View] --> ViewModel[ViewModel] ViewModel <--> Model[Model] </pre>

4.3.1.1.MVC 架构图填空（次重点★★★★☆）

这里有可能考你架构图填空，如下两个架构图是重点。第一个是 MVC 架构图。这里凯恩提一句，如图所示，View 和 Model 之间存在“状态查询”的交互关系。View 需要从 Model 获取数据来进行展示。但在一些设计规范严格的场景下，为保证架构清晰和代码可维护性、可扩展性，也可能要求 View 通过 Controller 间接获取 Model 数据。假如考试出填空，以下表为准。

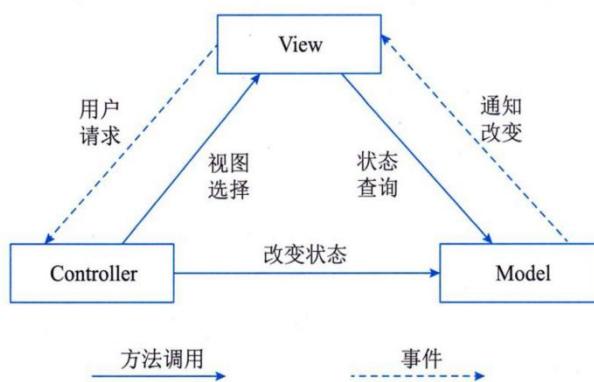
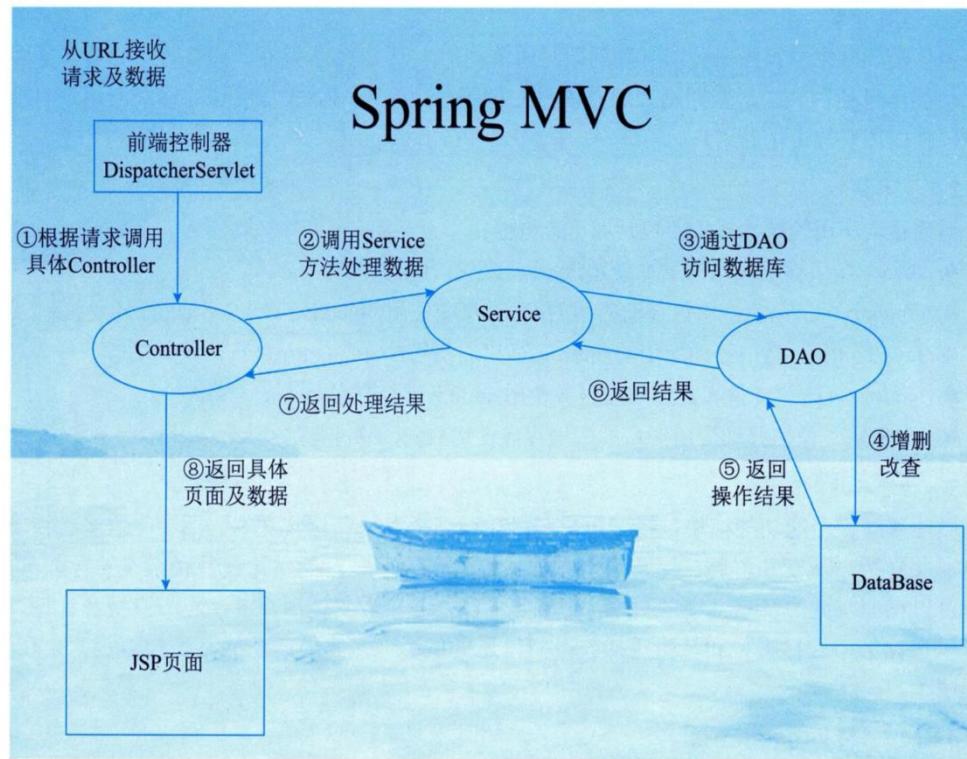


图 18-4 MVC 架构示意图

第二个是 SpringMVC 框架填图，这里涉及到 Controller、Service 和 DAO，横线上的内容你要填写的出来。



(1) 接收请求：前端控制器 DispatcherServlet 接收从 URL 来的请求及数据。它是请求进入应用的入口，负责后续的分发调度。

(2) 请求分发：DispatcherServlet 根据请求信息调用具体的 Controller。Controller 接收请求，处理用户交互相关逻辑，并决定调用对应的 Service 处理业务。

(3) 业务处理：Controller 调用 Service 方法。Service 层封装核心业务逻辑，会调用 DAO 访问数据库，完成如增删改查等操作。

(4) 数据访问：DAO 层负责与数据库交互，执行 SQL 语句实现数据的增删改查，然后将操作结果返回给 Service 层。

(5) 结果返回：Service 处理完业务逻辑后，将结果返回给 Controller。

(6) 视图渲染：Controller 接收结果后，选择合适的视图（如 JSP 页面），将数据填充到视图中，最后返回具体页面及数据给客户端展示。

4.3.1.2.MVP 架构图填空 (次重点★★★☆☆)

MVP 如之前表格总结，下图是教材给出的架构示意图，更新、查询、修改的操作你要写出来。

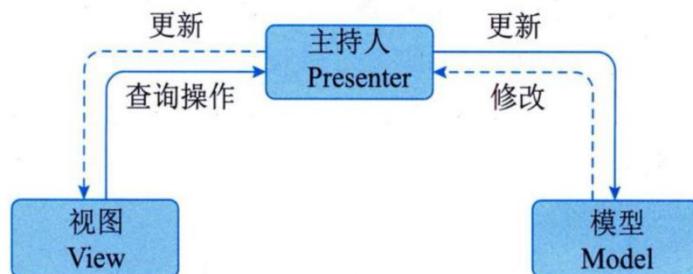


图 18-6 MVP 架构示意图

4.3.1.3.MVVM 架构图填空 (次重点★★★☆☆)

MVVM 如之前表格总结，下图是教材给出的架构示意图，更新、绑定的操作你要写出来。

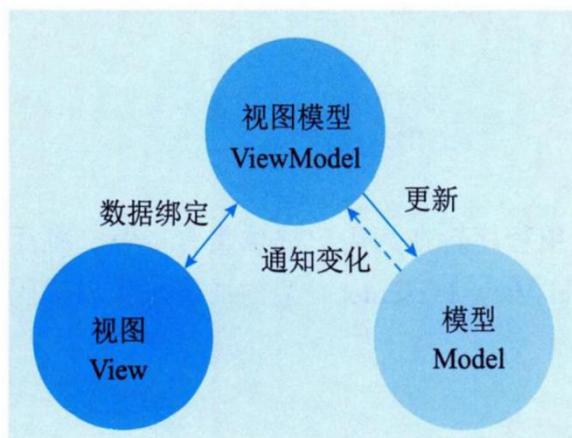


图 18-7 MVVM 架构示意图

这里单独把 Vue 的 MVVM 实现展开进行描述，下图 Observer, Dep, Watcher, Updater, Compile 的作用要清楚。

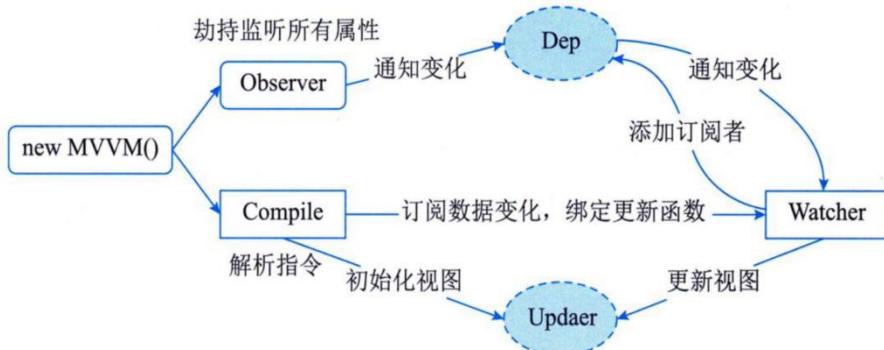


图 18-8 VUE 中的 MVVM 思想

指令解析与视图初始化：Compile（编译）像个小助手，会去解析写在模板里的指令，比如 v-bind v-if 这些，然后把页面初始状态展示出来。

数据监听：Observer（观察者）像个小哨兵，一旦属性有变化，它就知道。

订阅与通知：Dep（依赖）像是个小广播站，Observer 发现数据变了就通知它。Watcher（订阅者）提前在 Dep “订阅” 了数据变化，Dep 收到通知后，就会去告诉 Watcher。

视图更新：Watcher 收到通知，知道数据变了，就赶紧找到 Updater（更新器），Updater 就会按照新的数据把页面视图更新，让页面展示最新样子。这样就实现了数据和视图的自动关联更新。

4.3.2.P2P 架构（次重点★★★☆☆）

P2P 架构，即对等网络架构，是一种分布式的网络结构，在这种架构中，网络中的节点（可以是计算机、手机等设备）地位平等，没有专门的服务器来控制整个网络，每个节点既可以作为客户端请求资源，也可以作为服务器提供资源。其优势包括去中心化、资源共享高效、可扩展性强及隐私保护较好等，但也存在网络管理复杂、安全性挑战大、性能不稳定等缺陷。

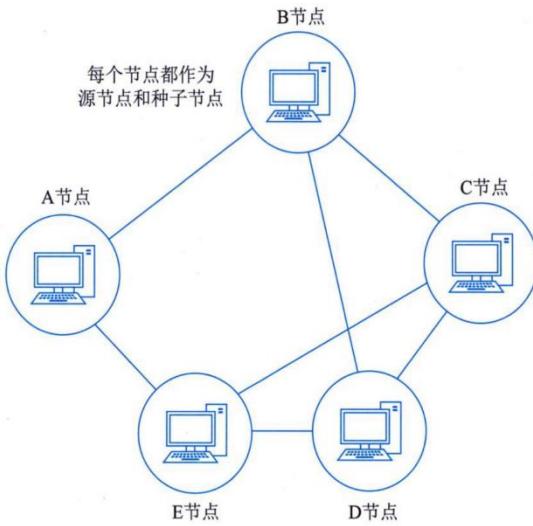


图 16-5 P2P 架构图

4.4. Web 应用部署（重点★★★★★）

这一章的内容不会直接考你，以往都是给定场景，以选词填空的方式来考察。

4.4.1. Web 应用部署原则（重点★★★★★）

原则类的系分爱考，就喜欢考这种概念，一般也是直接让你默写。简记：管理-安装-制度--改正。这里的底层逻辑就慎重交付，慎重体现在交付的节奏，交付的质量，建立反馈闭环等。

原则	具体内容
管理客户期望	与客户有效沟通，不轻易做可能完不成的承诺，利用敏捷方法便于增量交付，为客户提供更早反馈以管理期望
安装测试交付包	在部署前，要进行完整测试
建立支持制度	交付前建立记录保持机制，对支持请求种类进行分类评估
先改再交付	不交付质量低下的增量，鼓励用户评价并收集反馈，调整开发计划并改正问题

4.5. 典型例题（超级重点★★★★★）

(凯恩模拟) 某公司开发一个在线教育平台，包含用户认证、课程管理、支付系统、直播课堂、推荐引擎等模块。项目初期采用单体架构，所有功能耦合部署。随着用户量增长，系统频繁出现响应延迟、扩展性不足等问题。团队计划重构系统，同时需要遵循 Web 设计原则并选择合适的架构模式。

问：在重构过程中，团队决定将用户认证模块与支付系统分离，并引入接口层实现依赖反转。请结合 Web 设计原则，说明该做法符合哪些原则，并解释其核心思想。

此题直接考察 Web 设计原则，前面提到的分离关注点、依赖反转、单一责任原则都可以用到。

答：符合原则分离关注点、依赖反转、单一责任原则。分离关注点：将用户认证与支付系统拆分，降低模块间耦合，便于独立维护。依赖反转：通过接口层使高层模块（如业务逻辑）依赖抽象接口，而非具体实现（如数据库），支持灵活替换底层技术。单一责任：每个模块仅负责特定功能（如用户认证或支付），减少变更风险。

问：团队面临两种架构选择：单体架构、微服务架构。针对在线教育平台的直播课堂模块，应选择哪种架构？请从扩展性、维护成本两个维度对比分析，并给出选择理由。

此题直接考察架构模式的对比，那么一般来说微服务的优势就是扩展，解耦，缺点是管理复杂度高；服务间通信带来一定的开销；业务中假如需要保持事务一致，会有一些挑战。

答：选择微服务架构。在扩展性上，相比单体架构，微服务架构可以通过增加服务实例支持水平扩展，非常适合在直播课堂的高并发场景。另外在微服务架构中，各个服务可以独立更新，独立部署，对整个系统的影响相比单体架构小的多。

5.上篇-大数据架构（新教材★★★★★）

系分新教材也新增了这一章的内容，并且要求更高了，凯恩把系分新增的部分也放进来作为补充。鉴于现在命题考察形式，凯恩这里重点梳理出来的对比、图表一定要好好背出来，拿下案例第一题就稳了。

5.1.系统架构原则（次重点★★★☆☆）

这一小节的内容是系分新教材的补充内容，每个原则都有具体的展开，需要有个印象。这里的原则是指假如我要设计一个大数据系统，要做到哪些事情。一般不会直接考察这里的概念，而是给出指定场景让你归类属于什么设计界原则。

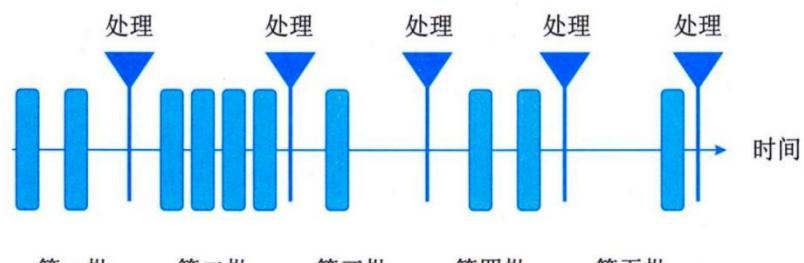
原则	要点	解释
可扩展	分布式环境	在分布式计算环境运行，分配任务、管理资源，实现负载均衡，支持横向扩展计算和存储资源
	模块化设计	将系统拆分为独立模块，便于设计、开发、维护，可添加或扩展模块
可管理	-	通过自动化管理、统一管理平台、可视化管理界面，实现高效管理监控，提升效率、可靠性等
数据安全	数据加密	应用于存储、传输、处理等，如用对称加密传输、非对称加密存储
	访问控制	通过用户名和密码等方式，为不同用户分配权限
	数据备份和恢复	定期备份数据，意外丢失时恢复
	审计日志	记录用户操作，用于审计跟踪、发现问题、满足监管
	持续监控	监控日志记录、访问模式等，及时发现解决安全问题
高性能	数据分区	将大数据集分块，分配给不同计算单元处理，提高并行度和扩展性
	分布式计算	将计算任务分布到多个节点，节点独立计算并合并结果，提高并行度和效率
	并行处理	将任务分成子任务，在多个计算单元同时执行

	内存计算	将数据存于内存计算，避免磁盘 I/O 和网络传输延迟，提高效率
高可用	数据冗余	将数据复制到多个节点或设备，故障时切换保证可用性
	自动故障转移	节点或设备故障时，自动切换到其他节点或设备，可通过 ZooKeeper 等工具实现
	负载均衡	将工作任务分配到多个节点或计算单元，平衡负载，可通过 Haproxy 等工具实现
稳定性	监控和预警	实时监控组件，异常时及时警报并处理
	性能调优	优化系统组件和配置参数，通过定期测试分析找到瓶颈优化
	故障排除	建立机制和团队，快速发现和解决故障
	升级和维护	及时更新组件和软件版本，定期维护修复，保证长期稳定可靠

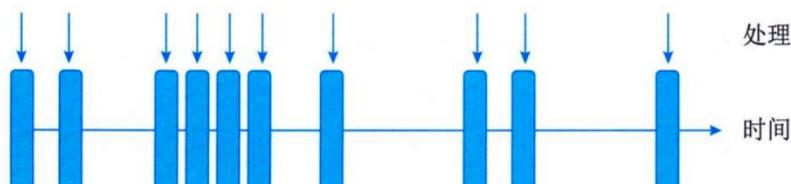
5.2. 批处理架构 VS 流处理架构 (重点★★★★★)

根据业务场景不同，我们对数据处理时效要求高，有时候要求低。这就引申出两种不同的数据处理模式，分别是批处理还有流式处理。此章节属于系分新增章节，比架构概括的好，特别是批处理和流处理的基本概念，优势和局限需要背出来，案例会考考他们的概念和相互的对比，这一节的内容必须熟悉。

架构类型	概念	优势	局限	应用场景
批处理架构	处理大规模数据批量任务，通常离线执行，含数据采集、存储、处理、输出模块；大数据集分组，按预定时间间隔或触发条件处理	可处理大规模数据，吞吐量高、稳定性好，能充分利用计算资源提升效率	实时性差，无法满足实时分析处理需求	对实时性要求不高的大规模数据处理场景



架构类型	概念	优势	局限	应用场景
流处理架构	数据连续、无限制流式处理，新数据到达即处理，由数据流、运算符组成，通过流处理器管理协调	实时处理数据，快速响应用户请求，适应数据快速变化	无法处理历史数据，可能漏数据，需考虑并发性和一致性等问题	实时数据分析、监控、推荐等场景

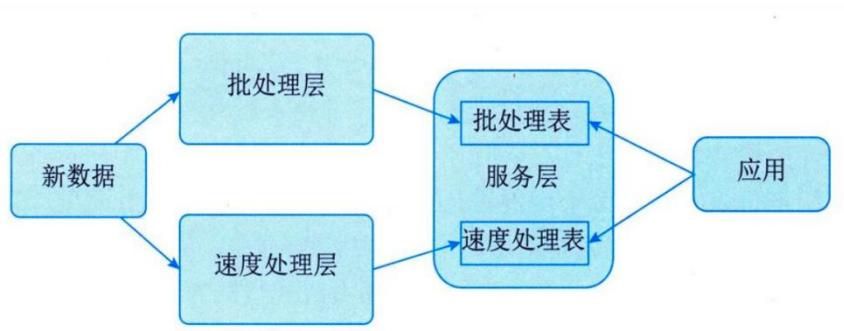


架构类型	概念	优势	局限	应用场景
混合架构	同时运用批处理和流处理，数据按需在两种处理模式间转换	兼具批处理和流处理优点，可处理历史和实时数据，快速响应用户请求	需在架构和数据处理上权衡，管理维护成本较高	既需处理历史数据又有实时数据处理需求的场景

5.3.Lambda 架构（重点★★★★★）

5.3.1.Lambda 分层介绍（重点★★★★★）

Lambda 架构的核心思想是：将批处理作业和实时流处理作业分离，各自独立运行，资源互相隔离。Lambda 架构可分解为三层，即批处理层、加速层（有的叫作速度层或者速度处理层）和服务层，如下图所示。



层次	概念	支撑技术	图例
批处理层	负责所有批处理操作，存储整个数据集并计算批处理视图	Hive、Spark-SQL、MapReduce 等批处理技术	<p>图19-6 批处理层结构：展示了批处理层（Batch Layer）与流处理层（Stream Layer）的交互。批处理层接收“全体数据集”并生成“结果视图（Batch View）”。流处理层接收“结果视图（Batch View）”并生成“结果视图（Batch View）”。</p>
加速层/速度层	使用流式计算技术实时处理当前数据，弥补批处理视图计算高延迟	Storm、Spark Streaming、Flink 等流处理技术	<p>图19-7 加速层结构：展示了批处理层（Batch Layer）与流处理层（Stream Layer）的交互。批处理层与流处理层并行处理，中间通过时间轴连接。</p>
服务层	以批处理层和加速层结果数据为基础，对外提供低延时数据查询和即席查询服务	关系型数据库等传统技术，或 Kylin、Presto、Impala、Druid 等大数据 OLAP 产品	<p>图19-8 服务层结构：展示了流处理层（Stream Layer）与批处理层（Batch Layer）的交互。流处理层进行“视图查询”，批处理层进行“视图查询”，两者结果合并。</p>

批处理层和加速层有羁绊的，常考它们的对比，分开设计的原因等，凯恩归纳为下表。

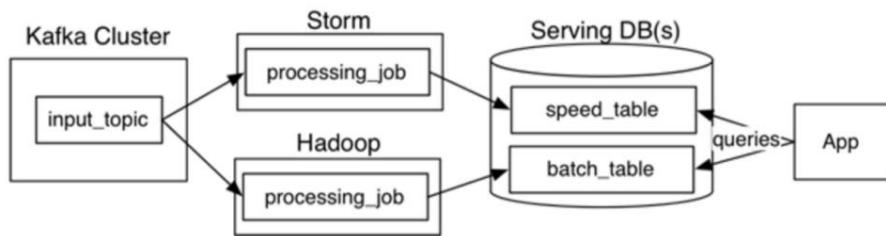
批处理层和加速层分开设计的原因	
容错性	加速层处理的数据持续写入批处理层。当批处理层重新计算的数据集涵盖了加速层处理的数据集后，可以修正加速层的一些错误
复杂性隔离	加速层运用增量算法处理实时数据，且其复杂性远高于批处理层，所以要进行隔离。
横向扩容	当系统面临的数据量或负载增大时，分层的设计可通过增加更多的机器资源来维持性能。

批处理层和加速层的对比		
对比内容	加速层	批处理层

处理目标	处理的数据是最近的增量数据流	处理的全体数据集
处理方式	接收到新数据时不断更新 Realtime View	根据全体离线数据集直接得到 Batch View

5.3.2.Lambda 架构实现（重点★★★★★）

上一节的分层结构是逻辑上的，那么如何落地，就是实现要谈的事情。这一节最简单的就是考你概念每个技术是做什么的，稍微复杂一点是技术之间的对比，或者给定场景让你填写具体的技术方案。



这是 Lambda 架构实现概览图，Kafka 集群的 input_topic 接收原始数据，Storm 进行实时处理、Hadoop 进行批量处理，处理后数据分别存入服务数据库的 speed_table 和 batch_table，App 通过向服务数据库发送查询获取数据用于业务。各个层次使用技术总结如下表所示。

层次	名称	功能	详细解释
批处理层	Hadoop	处理大规模数据集	Hadoop 是一个开源的分布式计算平台，主要用于批量数据处理。图中的 processing_job 代表在 Hadoop 上执行的作业，它也从 Kafka 的 input_topic 读取数据，不过是进行批量处理。Hadoop 适合处理大规模数据集，对处理时间要求相对宽松的场景。
加速层	Spark 或 Storm	提供快速数据处理能力，适合迭代计算的数据挖掘和机器学习任务	Spark 和 Storm 均为分布式计算框架：Spark 基于批处理，通过微批实现准实时（秒级延迟），擅长处理大规模数据、复杂计算（如机器学习、图分析）及批流一体任务，适合离线分析、日志处理等场景；Storm 是纯流处理框架，逐条实时处理数据（亚秒级延迟），强调低延迟与消息可靠性，适用于实时监控、支付系统、事件驱动架构（如 IoT）等需立即响应的场景。Spark 侧重吞吐量与复杂逻辑，Storm 聚焦极致实时性。

服务层	HBase 或 Cassandra	提供实时的数据访问和更新	HBase 是 Apache Hadoop 生态中的分布式列存储 NoSQL 数据库，基于 HDFS 构建，支持海量数据的高并发随机读写，适合实时查询、日志存储和时序数据场景；Cassandra 是高性能分布式宽列数据库，支持高吞吐量读写和复杂数据模型，常用于社交平台动态流、实时推荐系统和金融交易记录存储。
服务层	Hive	创建可查询的视图，方便数据查询和分析	Hive 构建在 Hadoop 之上，提供了类似于 SQL 的查询语言 HiveQL，能将 SQL 语句转换为 Map - Reduce 任务在 Hadoop 上运行，使得熟悉 SQL 的用户可以方便地对存储在 Hadoop 中的大规模数据进行查询和分析

5.3.3.Lambda 架构优缺点（重点★★★★★）

Lambda 架构的优缺点也是案例喜欢考察的内容。

优劣	描述	例子
优点	容错性好	Lambda 架构为大数据系统提供了更友好的容错能力，一旦发生错误，我们可以修复算法或从头开始重新计算视图
	查询灵活度高	数据存储多样性：批、实时处理结果存于不同存储系统，满足不同查询需求；多种查询接口支持：服务层提供 SQL、RESTful API 等接口，结合可视化工具展示结果
	易伸缩	分布式计算与存储：批、实时处理框架基于分布式架构，可按需添加节点；弹性资源调度：框架配备调度器，自动分配管理资源，依据任务需求动态调整
	易扩展	分层架构解耦性：分层设计，各层职责明确，扩展功能不影响其他层；插件化与接口化设计：各层组件采用插件、接口设计，便于集成新组件。书本上只从视图添加角度来讲，有点单薄。
缺点	全场景覆盖带来编码开销：技术栈复杂，开发维护成本高，代码重复；针对具体场景重新离线训练益处不大（针对推荐系统）：时间资源浪费，模型更新滞后（针对推荐系统）；重新部署和迁移成本高：环境配置复杂，数据迁移有一致性、速度等挑战	

5.4.Kappa 架构（重点★★★★★）

前面凯恩提到了 Lambda 架构的缺点，Kappa 在 Lambda 的基础上进行了优化，删除了

批处理层，将数据通道以消息队列进行替代。因此对于 Kappa 架构来说，依旧以流处理为主，数据在数据湖层面进行了存储，当需要进行离线分析或者再次计算的时候，则将数据湖的数据再次经过消息队列重播一次则可。核心组件包括：

核心组件	内容
消息传输层	使用 Kafka 等消息队列实现高吞吐、低延迟的数据传输，支持数据持久化和重放。
流处理层	采用 Flink、Storm 等框架进行实时计算，生成低延迟结果并输出至服务层。
数据存储	处理后的数据存储于 HBase、Cassandra 等高性能存储系统，支持快速查询

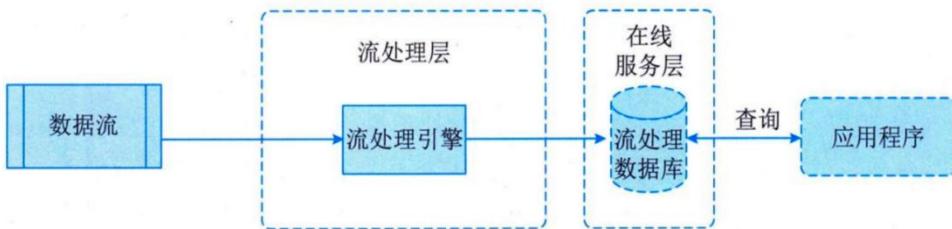


图 19-4 Kappa 架构图

5.4.1.Kappa 架构的优缺点（重点★★★★★）

Kappa 架构的优缺点也是案例喜欢考察的内容，需要加以记忆。

优劣	描述	例子
优点	将实时和离线代码统一，方便维护，统一数据口径	使用 Kappa 架构处理订单数据，开发人员无需分别维护实时和离线两套代码，能更高效地进行功能迭代和问题排查，避免因数据不一致导致的分析偏差。
缺点	消息中间件缓存的数据量和回溯数据有性能瓶颈	当需要回溯过去几个月的消息数据进行分析时，由于消息中间件缓存数据量过大，可能导致回溯操作缓慢甚至超时，影响数据分析的效率。

	实时数据处理时，大量不同实时流关联易因数据顺序问题导致数据丢失	在交通流量监控系统中，有车辆位置流、交通信号灯状态流等多个实时流数据。当关联这些数据计算车辆在不同信号灯状态下的通行情况时，若数据到达顺序混乱，可能会遗漏某些车辆在特定信号灯状态下的记录。
	抛弃离线数据处理模块，同时抛弃离线计算更稳定可靠的特点	采用 Kappa 架构后，过度依赖实时计算，在网络波动或实时计算系统出现短暂故障时，可能无法及时、准确地识别风险。

5.4.2.Lambda VS Kappa (超级重点★★★★★)

架构系分爱考概念对比，下面的表格内容需要熟记，23 年架构已经考过挖空填空题。

对比内容	Lambda 架构	Kappa 架构
复杂度与开发、维护成本	需要维护两套系统（引擎），复杂度高，开发、维护成本高	只需要维护一套系统（引擎），复杂度低，开发、维护成本低
计算开销	需要一直运行批处理和实时计算，计算开销大	必要时进行全量计算，计算开销相对较小
实时性	满足实时性（这个没错）	满足实时性
历史数据处理能力	批式全量处理，吞吐量巨大，历史数据处理能力强	流式全量处理，吞吐量相对较低，历史数据处理能力相对较弱

5.5.IOTA 架构 (次重点★★★☆☆)

系分新增的内容，了解即可，IOTA 的主要特点是引入边缘计算和统一数据模型。

IOTA 架构是一种新兴的大数据处理系统架构，它强调数据流的连续性和一致性。IOTA 的整体思路是设定标准数据模型，通过边缘计算技术把所有的计算过程分散在数据产生、计算和查询过程当中，以统一的数据模型贯穿始终，从而提高整体的计算效率，同时满足计算的需

要，可以使用各种即时查询来查询底层数据。下图是 IOTA 的基本架构。

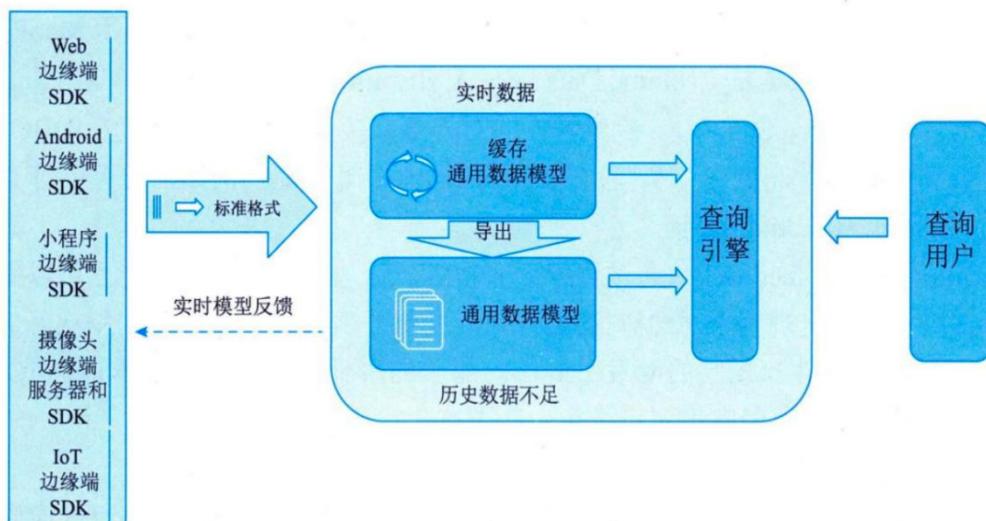


图 19-5 IOTA 架构图

书本上讲的很简单，看凯恩这里的补充。

统一数据模型（Common Data Model）。IOTA 架构通过定义“主-谓-宾”或“对象-事件”等抽象数据模型，贯穿数据采集、处理、存储全流程，确保从 SDK 到查询引擎的数据一致性。例如，用户行为数据可统一表示为“X 用户 - 事件 1 - A 页面（时间戳）”。

边缘计算与实时处理。在设备端（Edge SDKs）完成数据校验、格式转换和边缘聚合计算，减少中心化服务器的压力。例如，智能 Wi-Fi 采集的数据在设备端直接转换为“用户 MAC 地址 - 出现 - 楼层（时间）”的统一模型。

分层存储与实时缓存。实时数据缓存区（Real-Time Data）：使用 Kudu 或 HBase 暂存最近几分钟数据，支持低延迟查询。历史数据沉浸区（Historical Data）：通过 Dumper 将实时数据合并至 HDFS 等存储系统，支持秒级复杂查询。

IOTA 的优缺点如下表所示。

类别	要点	具体说明
优点	简化架构复杂度	删除 Lambda 架构的批处理层，仅需维护流式处理逻辑，降低开发和运维成本。

	低延迟处理	边缘计算 (Edge SDKs) 减少中心化处理步骤，支持毫秒级实时响应。
	统一数据模型	通过“主 - 谓 - 宾”模型统一数据格式，避免批流计算结果不一致问题，提升数据一致性。
缺点	历史分析性能瓶颈	依赖流式重放全量数据日志，超大规模历史数据分析效率较低，难以支持复杂离线计算。
	开发与调试难度高	需处理数据乱序、延迟、幂等性等问题，业务逻辑复用性低，开发复杂度较高。
	存储资源消耗大	需长期保留全量数据日志以支持重放，存储成本显著高于仅保留结果的 Lambda 架构。

5.6. 大数据系统开发（重点★★★★★）

这里最关键的是存储、管理和处理，这里涉及到大量的技术选型，反观分析和部署不太重要，这里略过，感兴趣的可以看一下书本相关内容。

5.6.1. 数据存储（重点★★★★★）

大数据数据存储流程包括数据采集、数据清洗、数据转换和数据存储。数据采集作为数据输入的关键部分，常用系统日志采集、ETL 工具采集、网络爬虫等方法，有离线、实时、互联网三种采集模式；数据清洗其基本流程包括数据分析、定义策略规则、搜寻确定错误实例、纠正错误、干净数据回流；数据转换目的是将数据转为可用格式，其步骤包括数据发现、数据映射、数据提取、代码生成、代码执行、数据审查。数据采集、数据清洗、数据转换不太重要，关键是数据存储方案要深入研究。大数据存储有两种方案可供选择：行存储和列存储，它们的对比必考，如下表所示。这里重点关注优缺点和各自的优化方式。

对比维度	行数据库	列数据库
------	------	------

存储单位	以一行记录为单位	以列数据集合（列族）为单位
读写过程	读取和写入都是从第一列开始，到最后一列结束	读取序列数据集中的一段或者全部数据；写入时，一行记录被拆分为多列，每列数据追加到对应列的末尾处
优点	写入效率高，保证数据完整性	大数据应用批量访问列数据时，对属性的访问比行存储方式快 50 - 100 倍；同类数据存储在一起有利于实施高效压缩算法；读取过程没有冗余，适合数据定长的大数据计算
缺点	数据读取有冗余现象，影响计算速度；不太适合数据量巨大、计算要求高的场景	缺乏数据完整性保证，写入效率低；写入次数明显比行存储多，加上磁头移动和定位时间，实际时间消耗更大
优化方式	优化存储格式，保证能够在内存快速删除冗余数据	多磁盘多线程并行读写（需要增加运营成本和修改软件）
典型技术	MySQL	Hadoop 的 HBase

5.6.2. 数据管理（次重点★★★★☆）

这一节涉及到元数据管理、数据关系图、数据安全和资源监控，案例很难出题目，了解即可。论文题目可能会考。

在大数据处理系统中，数据管理涵盖元数据管理、数据关系图谱、数据安全和资源监控。元数据即描述数据的数据，为系统数据提供上下文；数据关系图谱记录数据关联和血缘关系，利用关联分析算法挖掘数据价值；数据安全由认证和授权构成，遵循从产生到销毁的一系列安全流程；借助大数据技术获取、收集、分析数据，预测信息趋势。

5.6.3. 数据处理（重点★★★★★）

从大数据处理系统的数据处理方式来看，将大数据处理中的处理方式分为三种类型，即离线数据处理、实时数据处理和图计算处理。它们三种必定放在一起对比来考你 9 分的概念题，下面表格的对比项你要做到心中有数。

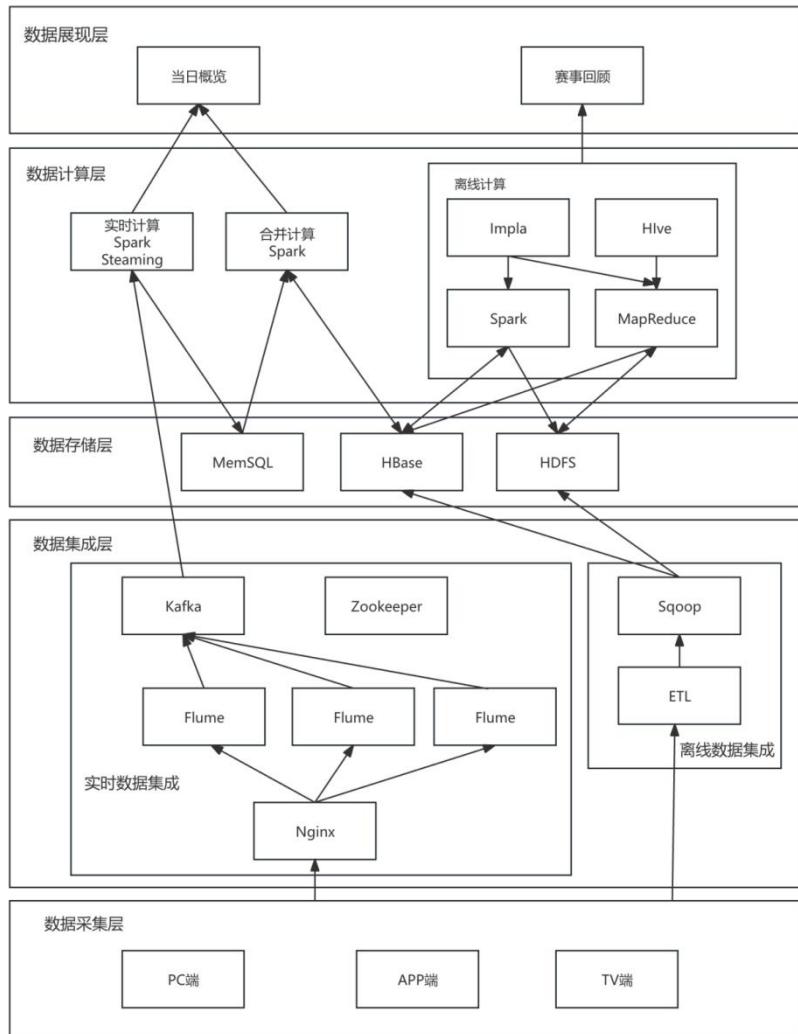
类型	实时计算处理	离线计算处理	图计算处理
数据特点	数据源源不断，无边界，需尽快处理，处理后数据量仍巨大	数据量巨大，有界，在计算前已完全到位且不会变化，保存时间长	数据以图结构表示，由顶点和边组成，可分为有向图、无向图等多种类型，顶点和边可包含属性
处理方式	采用流处理方式，数据能在秒级或毫秒级响应完成数据处理和展现	采用批处理方式，先让整体数据通过一个阶段处理，得到结果后再送下一阶段	使用专门的图算法进行处理，如路径搜索算法、中心算法、社群发现算法等
技术架构	Flume+kafka+Storm/Spark +Hbase/Redis 等	Hadoop 全家桶，计算完成的数据如需存储，直接存入 Hive，然后用 Hive 进行展现	在 Spark 框架下，主要由 Spark GraphX 提供图计算功能
优点	时效性强，能实时获取直观数据	数据处理技术成熟，能处理大量数据，结果准确性高，具有可持续性，数据存储后可断网处理	能有效处理复杂的关联关系，发现隐藏在数据中的结构和模式
缺点	需要不停计算数据，集群资源消耗大，数据周期短	时效性差，无法实时处理和展现结果	对图数据结构的理解和处理要求较高，算法实现相对复杂
应用场景	实时数仓、实时数据分析、流上机器学习等	数据仓库、搜索与检索、数据分析等	公安系统（打击犯罪团伙等）、银行金融领域（金融欺诈、信用卡盗刷、理财产品推荐等）

在数据处理领域，“有界”是指数据有明确的开始和结束，其规模在处理之前是已知的，或者说数据的范围是可以确定的，反之，“无界”指数据没有明确的边界或限制。

5.7.典型架构（重点★★★★★）

这里有三个典型架构，23 年出过挖空填空，目的是看你是否熟悉这些技术的具体应用。因此属于非常重点的精华章节，必须要看熟。

5.7.1.某网奥运（重点★★★★★）

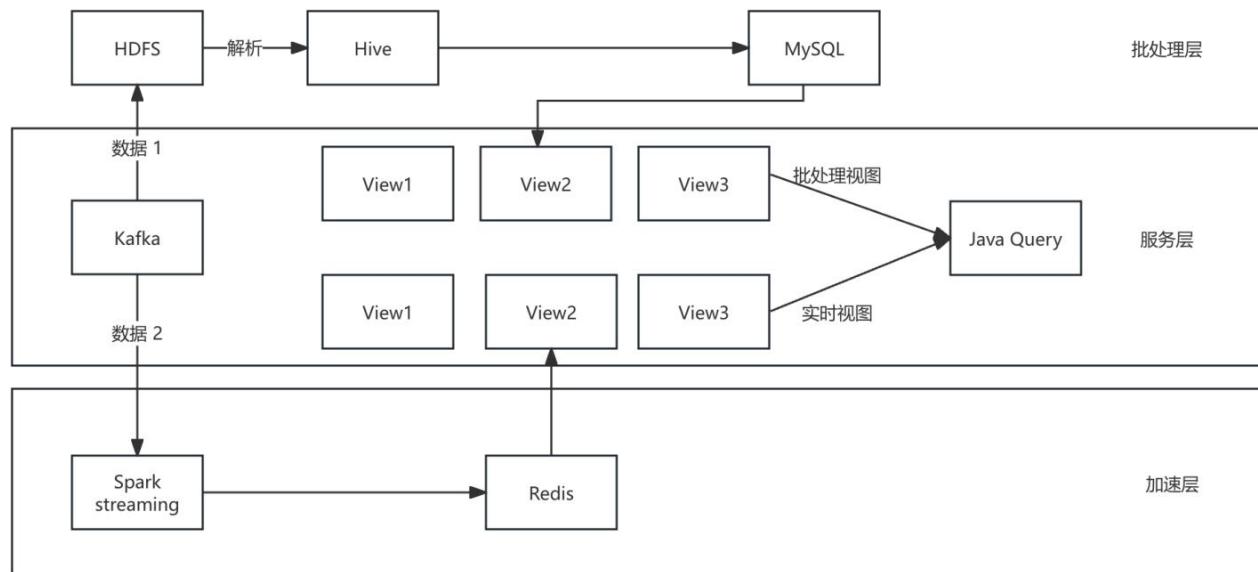


数据从 PC 端、APP 端、TV 端等来源，经数据采集层的 Flume、Kafka 等实时采集组件以及 Sqoop、ETL 等离线采集工具汇聚，存储于 MemSQL、HBase、HDFS 等存储层；计算层中，Spark Streaming 等负责实时计算，Impala、Hive 等进行离线计算；最终在数据展现层以日志展现、实时展现等形式呈现处理结果。此图中所涉及的技术可以归纳为下表，每个技术做什么的必须知道，选词填空中常见。

层级	涉及技术	用处说明
数据采集层	PC 端、APP 端、TV 端	作为数据来源，产生用户操作、业务等原始数据，是数据流程的起点。
数据集成层	Flume	实时收集、聚合、传输分散日志数据，高效传输至 Kafka。
	Nginx	负载均衡，优化数据采集节点流量分配，保障数据稳定收集。
	Kafka	分布式消息队列，缓存实时数据，解耦生产与消费，支持高吞吐量流式处理。
	Zookeeper	为 Kafka 等组件提供协调服务，管理集群节点状态与配置信息。
	ETL	对离线数据提取、转换、加载，清洗并转换为适合分析的格式。
	Sqoop	在关系型数据库与 Hadoop 生态间批量导入 / 导出数据，整合异构离线数据源。
数据存储层	MemSQL	内存数据库，高速读写，存储实时计算结果或高频访问热数据。
	HBase	分布式 NoSQL 数据库，存储海量半结构化 / 非结构化数据，支持高并发随机读写。
	HDFS	分布式文件系统，存储大规模离线数据，具备高容错性和扩展性。
数据计算层	Spark Streaming	流计算框架，处理实时数据流，实现实时统计、分析。
	Spark	通用分布式计算框架，处理批量数据合并计算，支持复杂数据处理逻辑。
	Impala	实时交互式查询引擎，配合 Hive 对 HDFS/HBase 数据快速 SQL 查询。

	Hive	将结构化数据映射为表，支持 HQL 查询，转换为 MapReduce 任务处理离线数据分析。
	MapReduce	离线批量计算模型，拆分任务为 Map 和 Reduce 阶段，处理大规模分布式计算。
数据展现层	当日概览、赛事回顾	将计算结果可视化呈现，供用户查看实时动态或历史数据，实现数据价值落地。

5.7.2. 某网络广告平台（重点★★★★★）

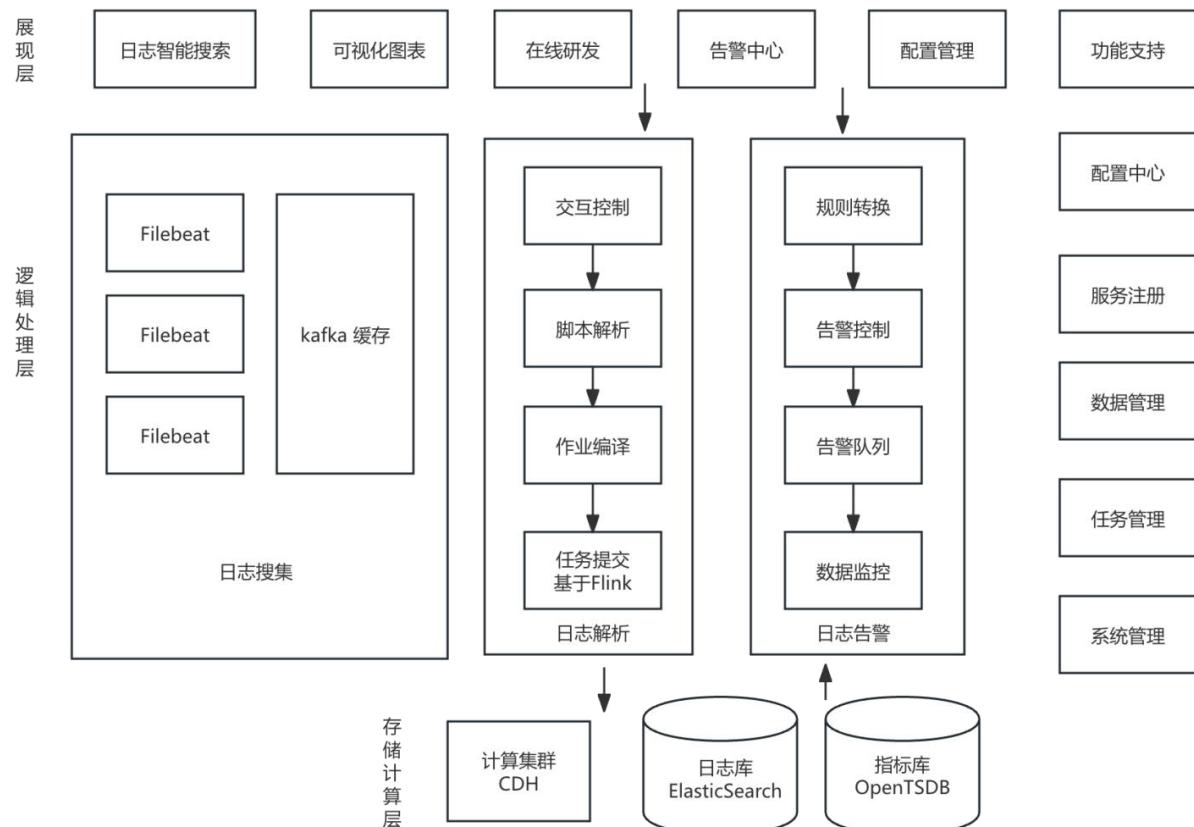


在批处理层，HDFS 存储原始数据，Hive 对其进行离线分析处理后存入 MySQL；服务层里，Kafka 作为消息队列传输数据，批处理视图和实时视图（View1 - View3）基于不同处理结果构建，通过 Java Query 获取数据；加速层中，Spark Streaming 进行实时数据处理，Redis 缓存处理结果以提升性能，各层组件协同满足业务的数据处理与查询需求。

层级	涉及技术	用处说明
批处理层	HDFS	分布式文件系统，用于存储大规模原始数据，为后续批处理提供数据基础。
	Hive	基于 HDFS 的数据仓库工具，通过 HQL 解析处理 HDFS 中的数据，实现批量数据的分析、转换。

	MySQL	关系型数据库，存储 Hive 处理后的结构化结果数据，供上层服务层查询调用。
服务层	Kafka	分布式消息队列，接收并缓存数据（数据 1、数据 2），解耦数据生产与消费，支持实时数据流处理。
	View1-View3	分为批处理视图和实时视图，对批处理层（如 MySQL）和加速层处理后的数据进行逻辑封装，方便上层统一查询。
	Java Query	业务查询接口，通过调用批处理视图和实时视图，实现数据的业务逻辑查询与展示。
加速层	Spark Streaming	流计算框架，实时处理 Kafka 中的数据（数据 2），进行实时计算、清洗或聚合。

5.7.3.某证券公司日志大数据系统（重点★★★★★）

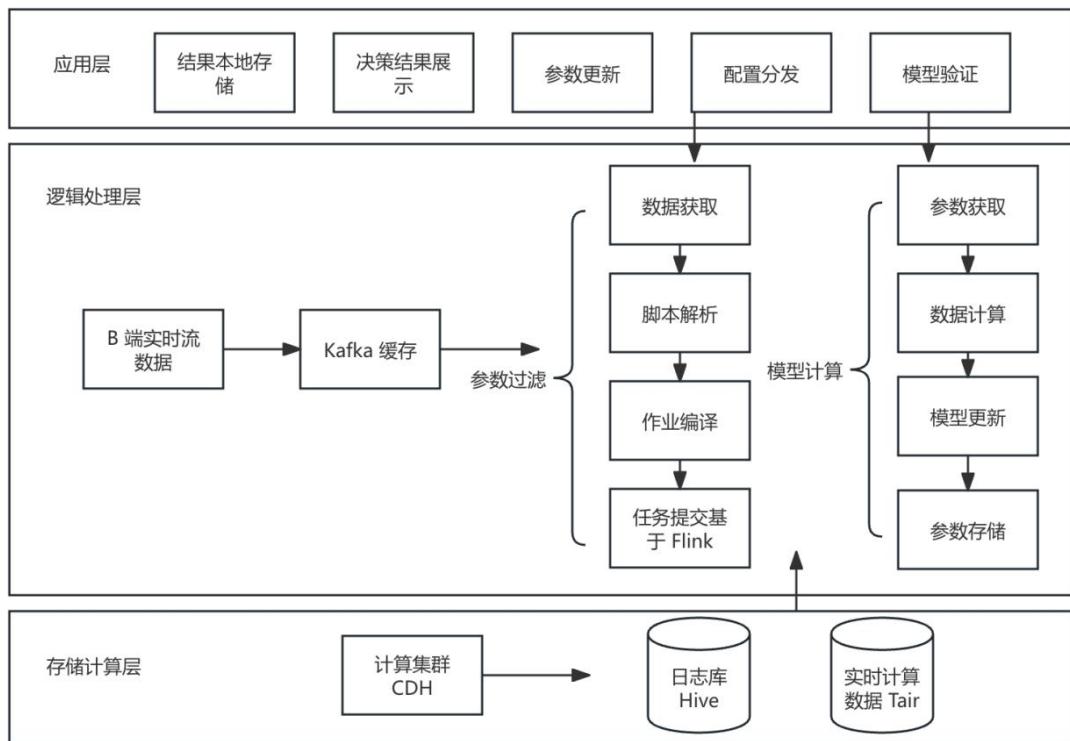


在逻辑处理层，Filebeat 负责日志搜集并暂存于 Kafka 缓存；搜集后的日志经交互控制、

脚本解析、作业编译等流程基于 Flink 进行日志解析，任务提交至计算集群 CDH，同时规则转换、告警控制等实现日志告警，数据分别存入日志库 ElasticSearch 和指标库 OpenTSDB。在展现层，提供日志智能搜索、可视化图表等功能，整个系统由配置中心、服务注册等进行管理支持。这里的关键是逻辑处理层和存储计算层，这里的技术梳理如下所示。

层级	涉及技术	用处说明
逻辑处理层	Filebeat	轻量级日志收集工具，收集不同数据源日志，传输至 Kafka。
	Kafka	缓存日志数据，解耦日志收集与处理，支持高吞吐量实时数据缓冲。
存储计算层	CDH	大数据计算集群，提供 Hadoop 生态组件，支持日志数据分布式存储与计算。
	ElasticSearch	作为日志库，存储处理后的日志，支持高效检索与分析。
	OpenTSDB	作为指标库，存储时间序列数据，适合高频更新的时序指标存储与查询。

5.7.4. 某电商智能决策大数据系统（重点★★★★★）



B 端实时流数据先经 Kafka 缓存，经参数过滤后，在逻辑处理层，一方面通过数据获取、脚本解析、作业编译，基于 Flink 提交任务；另一方面经参数获取、数据计算等实现模型计算及更新。存储计算层中，计算集群 CDH 配合日志库 Hive、实时计算数据 Tair 进行存储计算。应用层提供结果本地存储、决策结果展示等功能。这个架构图的逻辑和上一个很像，唯一区别是存储计算层的技术这里选的是 Hive。

层级	技术	用处说明
逻辑处理层	Kafka	作为消息队列，缓存 B 端实时流数据，解耦数据生产与消费，支持高吞吐量数据缓冲。
	Flink	基于 Flink 提交作业任务，处理实时流数据，实现数据的实时计算与作业执行。
存储计算层	CDH	提供大数据计算集群环境，支持分布式存储与计算，是底层数据处理的基础平台。
	Hive	作为日志库，存储处理后的日志数据，支持批量数据的存储、查询与分析。

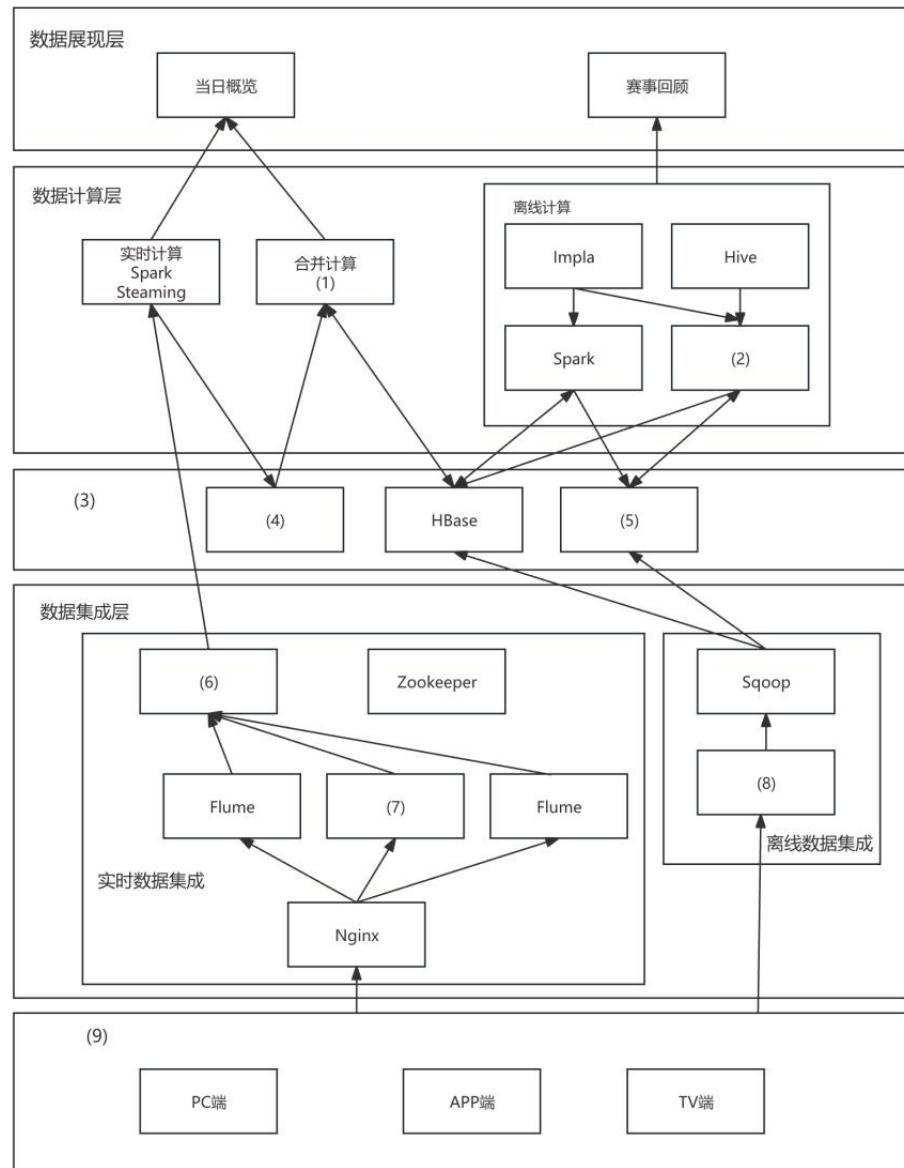
	Tair	存储实时计算数据，利用其高效读写特性，支持实时计算结果的快速存储与获取。
--	------	--------------------------------------

5.8. 典型例题（重点★★★★★）

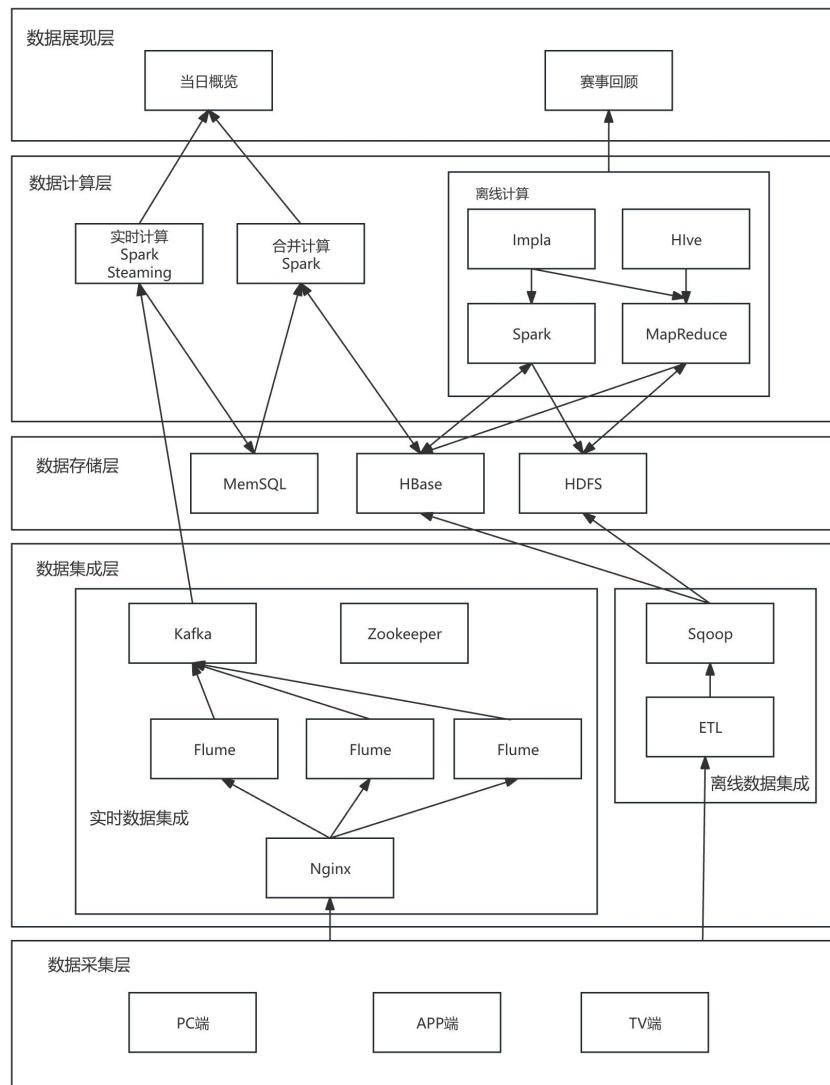
(23 年架构真题) 某网作为某电视台在互联网上的大型门户入口，某一年成为某奥运会中国大陆地区的特权转播商，独家全程直播了某奥运会全部的赛事，积累了庞大稳定的用户群，这些用户在使用各类服务过程中产生了大量数据，对这些海量数据进行分析与挖掘，将会对节目的传播及商业模式变现起到重要的作用。该奥运期间需要对增量数据在当日概览和赛事回顾两个层面上进行分析。

问：下图给出了某网奥运的大数据架构图，请根据下面的 (a)~(n) 的相关技术，判断这些技术属于架构图的那个部分，补充完善下图 1 的 (1) - (9) 的空白处。

(a) Nginx;(b) Hbase;(c) Spark Streaming(d) Spark;(e) MapReduce;(f) ETL;(g) MemSQL; (h) HDFS; (i)Sqoop; (j) Flume ; (k) 数据存储层； (l) kafka; (m) 业务逻辑层； (n) 数据采集层



答: (1) d (2) e (3) k (4) g (5) h (6) l (7) j (8) f (9) n



6.上篇-云原生架构设计（超级重点★★★★★）

这一章属于比较重点的章节，云原生微服务也放在这里进行讨论。目前案例还没出过真题。特别这里的概念类的简述和对比值得关注。书上说了很多，其实用云原生的主要目的就是减少运维成本，用金钱换省事，所以它常常和运维联系在一起，历年真题论文中考过 Devops 相关内容。

6.1.云原生架构原则（重点★★★★★）

云原生架构的必要条件如下所示。这里唯一需要注意的是弹性原则和韧性原则的区别。乍看好像都是为了针对业务来做资源伸缩，保障系统稳定运行。实际上弹性侧重“资源动态适配业务需求”，韧性侧重“故障场景下的系统保护”，记住这一点即可，防止案例让你比较。

原则	描述
服务化原则	项目规模大时拆分服务，提升复用性与流量治理能力
弹性原则	系统自动伸缩，适配业务需求
可观测性原则	通过日志、链路跟踪等掌握系统状态
韧性原则	抵御软硬件异常，保障系统稳定
全流程自动化原则	自动化交付提升效率、降低风险
零信任原则	基于身份访问控制，替代传统网络边界安全
持续演进原则	架构适应技术与业务变化，渐进式升级

6.2.云原生架构模式（重点★★★★★）

这里主要提到 7 种云原生架构模式。这些架构在实际场景中往往组合使用。案例中一般会给出实际场景，让你判断属于哪种模式，不太会让你默写概念。看到名字你要想到它们的内

涵。这里多说一句，下表所列的架构模式，极有可能在论文中出现。

架构模式	核心内容
服务化架构模式	以应用模块划分软件，通过接口契约、标准协议定义业务关系，结合 DDD、TDD、容器化部署提升代码质量与迭代速度，典型模式为微服务、小服务
Mesh 化架构模式	将中间件框架（RPC、缓存等）从业务进程分离，业务进程仅保留薄 Client，中间件逻辑由 Mesh 进程完成
Serverless 模式	开发者无需关心应用运行环境，云自动处理应用启动、调度、关闭，业务流量 / 事件触发处理
存储计算分离模式	分布式环境中，将暂态数据、持久数据采用云服务保存，有状态场景通过时间日志 + 快照恢复服务
分布式事务模式	针对多数据源场景，提供多种事务模式：XA 模式（强一致，性能差）；基于消息最终一致性（高性能，通用性有限）；TCC 模式（应用层控制，侵入性强）；SAGA 模式（补偿事务，开发维护成本高）；SEATA AT 模式（高性能、无代码工作量，有场景限制）
可观测架构	通过 Logging（多级别日志）、Tracing（请求链路跟踪）、Metrics（系统量化度量）实现可观测性，定义组件 SLO（并发度、耗时等）
事件驱动架构	基于事件集成应用 / 组件，事件含 schema 可校验，具备 QoS 保障与失败响应机制，用于服务解耦、数据变化通知、事件流处理等

6.3. 不好的实践（重点★★★★★）

假如说云原生架构是正面教材，那么你不照着他的原则来就是反面教材，也就是所谓的反模式、不好的实践或者说糟糕的做法，书本提到了三点。这里最简单的考法就是列出下表，让你简答填空，默写概念。

云原生架构反模式	问题描述	解决方案
庞大的单体应用	缺乏依赖隔离，存在代码耦合、模块间接口缺乏治理、不同模块开发发布进度难以协调、单个模块不稳定影响整个应用等问题	通过服务化进行适度拆分，梳理聚合根，明确服务模块边界和模块间接口定义，使组织关系和架构关系匹配

单体应用“硬拆”为微服务	过度服务化拆分会致使新架构与组织能力不匹配，影响架构升级效果。具体表现有小规模软件过度拆分、服务间数据依赖、服务拆分导致性能下降等	合理评估拆分粒度，充分考量组织能力与业务实际需求，避免过度拆分，优化服务间的数据交互设计，提升整体性能
缺乏自动化能力的微服务	软件规模增大时，人工处理开发测试运维等工作会造成功交付时间变长、风险提升、运维成本增加等问题	建立完善的自动化能力，涵盖自动化测试、发布、环境管理等，以适应复杂度提升的需求

6.4. 容器技术（重点★★★★★）

容器作为标准化软件单元，它将应用及其所有依赖项打包，使应用不再受环境限制，在不同计算环境间快速、可靠地运行。容器技术只要出题肯定会和虚拟机进行比较。另外单纯的容器和容器编排的区别也是重点考察的内容。

6.4.1. 容器和虚拟机对比（重点★★★★★）

容器和虚拟机的对比如下表所示，其中标注灰色的运行原理，资源隔离方式，启动速度，资源占用，隔离性的不同之处都要给我背出来（不需要一字不差）。

对比维度	容器	虚拟机
运行原理	基于操作系统层面的虚拟化，共享宿主机内核，直接运行在操作系统之上	通过硬件虚拟化技术，模拟出完整的硬件环境，运行独立的操作系统
资源隔离方式	利用 cgroups 进行资源限制（如 CPU、内存、磁盘 I/O 等），通过 namespace 实现命名空间隔离（如进程、网络、文件系统等），不同容器共享内核但彼此隔离	依靠硬件虚拟化技术，在不同虚拟机实例间实现完全的资源隔离，每个虚拟机有独立的内核和硬件资源
启动速度	启动非常快，一般在秒级。因为无需启动完整操作系统，仅需加载应用及其依赖	启动相对较慢，通常需要数十秒到数分钟，要加载整个操作系统

资源占用	占用资源少，仅需运行应用所需的资源，因为共享宿主机内核	资源占用多，每个虚拟机都有独立的操作系统和完整硬件资源模拟，开销大
应用移植性	移植性好，将应用及其依赖打包在镜像中，环境一致性高，可在不同支持容器的环境中快速部署	移植有一定难度，受虚拟硬件环境和操作系统影响，迁移时需考虑兼容性
隔离性	隔离性相对较弱，虽能实现进程、网络等隔离，但共享内核，存在一定安全风险（如内核漏洞可能影响多个容器）	隔离性强，每个虚拟机相互独立，一个虚拟机出现问题基本不影响其他虚拟机
管理复杂度	相对简单，主要管理容器镜像、容器生命周期（创建、启动、停止、删除等），通过编排工具可实现集群管理	管理复杂，需管理多个虚拟机的操作系统安装、配置、补丁更新等，以及虚拟硬件资源的分配和管理

6.4.2. 容器和容器编排技术（重点★★★★★）

容器和容器编排技术并非同一类事物的对比，容器是一种轻量级、可移植的应用打包和运行环境，而容器编排技术是用于管理和协调多个容器的技术，它们之间对比如下。它们的定义是最关键的部分，一定要记忆。

对比维度	容器	容器编排技术
定义	一种将应用及其依赖项打包成独立单元的技术，可在不同环境中运行，共享宿主机内核	用于自动化部署、管理和扩展容器化应用程序的技术，能管理多个容器组成的集群
核心功能	隔离应用运行环境，实现应用的快速部署和迁移	容器集群管理、资源调度、服务发现、负载均衡、自动伸缩、故障恢复等
代表工具	Docker	Kubernetes (k8s)
部署难度	部署单个容器相对简单，通过拉取镜像并启动容器即可	部署和配置容器编排系统较为复杂，涉及多个组件的安装、配置和集群初始化

适用场景	适合部署单一、简单的应用，或作为微服务架构中单个服务的运行载体	适用于大规模、复杂的容器化应用集群管理，如大型互联网应用、企业级分布式系统，这些系统需要高效的资源调度、自动伸缩和服务治理等功能
管理粒度	主要管理单个容器的生命周期，如创建、启动、停止、删除，以及容器内应用的运行状态	从集群层面管理多个容器，包括容器的分组、调度到不同节点、监控容器的健康状态、动态调整容器数量等

6.4.3. 容器运维指令（重点★★★★★）

有一年系分案例考过 docker 指令，所以还是要有所了解，至少有个印象。

Docker 常用指令	
docker ps	查看当前所有运行的容器
docker ps -a	查看所有容器（包括停止的）
docker run [选项] [镜像名称] [命令] [参数]	启动一个容器
docker stop [容器 ID 或名称]	停止一个运行中的容器
docker kill [容器 ID 或名称]	强制停止一个运行中的容器
docker restart [容器 ID 或名称]	重启容器
docker rm [容器 ID 或名称]	删除一个容器
docker rm \$(docker ps -a -q)	删除所有容器
docker logs [容器 ID 或名称]	查看容器的日志
docker exec -it [容器 ID 或名称] /bin/bash	进入容器内部
docker build [选项] 路径	构建镜像
docker images	查看镜像
docker rmi [镜像 ID 或名称]	删除镜像
docker pull [镜像名称]	拉取镜像
docker push [镜像名称]	推送镜像到仓库
docker stats	查看 Docker 容器的资源使用情况
docker info	查看 Docker 的信息

docker --help	查看 Docker 的帮助
---------------	---------------

Kubernetes (k8s) 常用指令。

指令	说明
kubectl get nodes	查看集群内节点信息
kubectl get pods	查看当前命名空间下的 Pod 列表
kubectl describe pod <pod-name>	查看指定 Pod 的详细信息，用于故障排查
kubectl delete pod <pod-name>	删除指定的 Pod
kubectl get services	查看当前命名空间下的服务列表
kubectl get deployments	查看当前命名空间下的 Deployment 列表
kubectl set image deployment/<deployment-name> <container-name>=<new-image>	更新 Deployment 中容器的镜像
kubectl logs <pod-name>	查看指定 Pod 的日志

6.4.4. 典型例题（重点★★★★★）

(23 年系统分析师) 随着嵌入式计算资源快速提升，容器技术 (Docker) 发挥重要作用，某公司对原有平台升级，公司将平台升级任务交给了张工，张工经过分析、调研，提出在嵌入式操作系统平台上采用容器技术的升级方案，但该方案引发了争议。

问：争论焦点是采用容器技术还是虚拟机 (VM) 技术。李工指出由于容器技术共享主机内核能像虚拟机一样完全隔离，系统存在安全问题；如果采用虚拟机技术除了满足需求外，还保证了系统的安全和稳定，会上领导根据系统升级的初衷选择了张工的升级方案，请用 300 字以内的文字说明容器技术和虚拟技术的含义，并简要论述公司领导采纳容器技术的原因。

这里实际就是讨论容器和虚拟机的区别，前面小节提到的表格这里可以派上用场。

答：容器技术是一种轻量级的隔离技术，多个容器共享主机内核，每个容器运行一个或一组应用，具备自己的文件系统、运行环境等，实现应用的快速部署和隔离。虚拟机技术通过虚

拟化软件在物理主机上模拟出多个独立的虚拟计算机，每个虚拟机有自己的操作系统、硬件资源，实现完全隔离。

容器技术相比虚拟机更加轻量化，资源占用少，启动速度快，能更高效利用硬件资源，满足系统升级中对资源利用优化的需求。且在当前技术发展下，安全问题可通过安全策略、安全工具等手段进行防护，在满足业务需求的同时，能以更低成本、更高效率完成部署，契合系统升级初衷。

问：下表给出了虚拟技术和容器技术的性能对比表，请根据下面的 (a) ~ (h) 的 8 个性能指标；判断这些指标属于哪类对比项，补充完善 3-1 的 (1) - (8) 的空白处。

(a) 分钟级、(b) 包含 GuestOS, G 量级以上、(c) 跨操作系统平台迁移、(d) CPU 与内存按核、按 G 分配 (e) 毫秒级、(f) Cgroups, 进程级别、(g) VM 伸缩, CPU/内存手动伸缩、(h) 实例自动伸缩、CPU 内存自动在线伸缩。

对比项	虚拟机技术	容器技术
镜像大小	①	仅包含运行的 Bin/Lib, M 量级
资源要求	②	CPU 与内存按单核、低于 G 量级分配
启动时间	③	④
可持续性	跨物理机迁移	⑤
弹性伸缩	⑥	⑦
隔离策略	操作系统、系统级别	⑧

此题考察虚拟机和容器技术的对比，本质上和第一问没有区别，见容器那一小节的讨论。

答：①b ②d ③a ④e ⑤c ⑥g ⑦h ⑧f

7.上篇-信息安全（超级重点★★★★★）

7.1.数据加密技术（超级重点★★★★★）

数据加密技术架构 2013 年考过一次真题，就是考你加密技术的概念，这里作为基本知识

还是要了解。

7.1.1. 对称加密概念（超级重点★★★★★）

对称加密是一种加密技术，它使用同一个密钥来加密和解密数据。也就是说，发送方使用密钥将明文加密成密文，接收方使用相同的密钥将密文还原为明文。其工作原理是基于某种数学算法，对原始数据进行特定的变换，使得只有拥有正确密钥的人才能还原数据。对称加密的优点是加密和解密速度快，效率高，适合对大量数据进行加密。

7.1.2. 非对称加密概念（超级重点★★★★★）

非对称加密则使用一对密钥，即公钥和私钥。公钥可以公开给任何人，用于加密数据；而私钥则由用户自己保密，用于解密数据。当发送方要向接收方发送数据时，使用接收方的公钥对数据进行加密，接收方收到密文后，使用自己的私钥进行解密。非对称加密的安全性基于数学难题，如 RSA 算法基于大整数分解难题，使得在不知道私钥的情况下，很难从密文还原出明文。它的主要优点是密钥管理方便，不需要像对称加密那样在通信双方之间安全地传递密钥。非对称加密常用于数字签名、密钥交换等场景。

7.1.2.1. 非对称加密的应用场景（超级重点★★★★★）

非对称加密的基本核心是“公钥加密私钥解密，也可以私钥加密公钥解密”，这两种方法衍生出了两种不同的场景。

私钥加密公钥解密。在数字签名场景中，通常是使用私钥对消息进行签名，然后将消息以及签名一起发送给接收方，接收方使用对应的公钥来验证签名，以确保消息的来源和完整性。这个时候传递的内容是明文。

公钥加密私钥解密。在加密场景中，一般是使用接收方的公钥对消息进行加密，然后接收方使用自己的私钥进行解密。这样可以保证只有拥有相应私钥的接收方能够解密并读取消息内容。这个时候传递的内容是加密的。这个你结合数字签名那一章来看会比较明白。

7.1.3. 对称加密和非对称加密的区别（超级重点★★★★★）

案例前面提到过，最喜欢考场景的技术选型还有技术之间的对比，那么对称加密和非对称加密特别在加密的速度和适用的数据规模上各有不同，看凯恩下表的总结。

对比项	对称加密	非对称加密
密钥数量	单一密钥	密钥对（公钥 + 私钥）
加密速度	快（适合大数据）	慢（适合小数据）
密钥管理	需安全共享	公钥公开，私钥保密
主要用途	数据加密	密钥交换、数字签名、身份验证
数学基础	替代 / 置换操作	数论难题（分解质因数、离散对数等）
典型算法	AES、DES、3DES	RSA、ECC、DH

7.1.4. 典型例题（超级重点★★★★★）

(13 年架构真题) 某软件公司拟开发一套信息安全支撑平台，为客户的局域网业务环境提供信息保护。该支撑平台的主要需求如下：(1) 为局域网业务环境提供用户身份鉴别与资源访问授权功能；(2) 为局域网环境中交换的网络数据提供加密保护；(3) 为服务器和终端机存储的敏感持久数据提供加密保护；(4) 保护的主要实体对象包括局域网内交换的网络数据包、文件服务器中的敏感数据文件、数据库服务器中的敏感关系数据和终端机用户存储的敏感数据文件；(5) 服务器中存储的敏感数据按安全管理员配置的权限访问；(6) 业务系统生成的单个敏感数据文件可能会达到数百兆的规模；(7) 终端机用户存储的敏感数据

为用户私有；（8）局域网业务环境的总用户数在 100 人以内。

问：在确定该支撑平台所采用的用户身份鉴别机制时，王工提出采用基于口令的简单认证机制，而李工则提出采用基于公钥体系的认证机制。项目组经过讨论，确定采用基于公钥体系的机制，请结合上述需求具体分析采用李工方案的原因。

这里实际上考察的就是非对称加密的概念：首先，公钥体系相对于基于口令的简单认证机制更安全可靠，因为口令容易受到猜测、暴力破解等攻击，而公钥体系利用非对称加密技术，安全性更高。其次，公钥体系可以提供更强的身份验证和数据传输加密保护，确保用户身份的安全性和数据的机密性。这个在介绍非对称加密的时候都已经提到。

答：（1）基于口令的认证方式实现简单，但由于口令复杂度及管理方面的原因，易受到认证攻击；而在基于公钥体系的认证方式中，由于其密钥机制的复杂性，同时在认证过程中私钥不在网络上传输，因此可以有效防止认证攻击，与基于口令的认证方式相比更为安全。

（2）按照需求描述，在完成用户身份鉴别后，需依据用户身份进一步对业务数据进行安全保护，且受保护数据中包含用户私有的终端机数据文件，在基于口令的认证方式中，用户口令为用户和认证服务器共享，没有用户独有的直接秘密信息，而在基于公钥的认证方式中，可基于用户私钥对私有数据进行加密保护，实现更加简便。

（3）基于公钥体系的认证方式协议和计算更加复杂，因此其计算复杂度要高于基于口令的认证方式，但业务环境的总用户数在 100 人以内，用户规模不大，运行环境又为局域网环境，因此基于公钥体系的认证方式可以满足平台效率要求。

7.1.5.数字签名（重点★★★★★）

数字签名（Digital Signature）是公钥基础设施（PKI）的核心组成部分。在 PKI 体系中，常见的要素包括数字证书、证书颁发机构（CA）、银行使用的安全 Key，以及 SSL/TLS 通

信等，而数字签名正是这些应用的技术基础。

7.1.5.1.Hash 算法与数字签名（重点★★★★★）

在理解数字签名之前，必须先掌握 Hash（散列/杂凑）算法。Hash 算法的主要特性包括：
易压缩性：可将任意长度的数据映射为固定长度的输出。单向性：从源数据得到哈希值容易，但无法由哈希值推回源数据。高灵敏性：输入数据发生微小变化，输出结果会有显著差异。抗碰撞性：不同输入得到相同哈希值的概率极低。

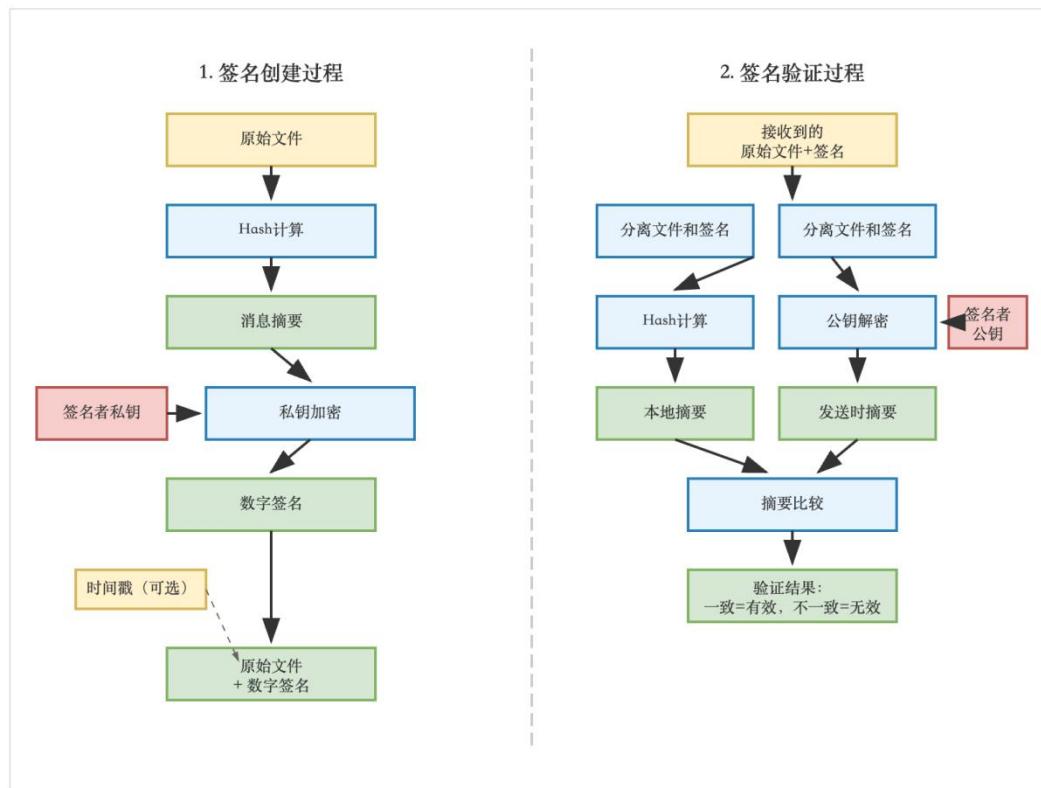
常用算法包括 SHA-1、SHA-256、SHA-384 以及我国的 SM3。由于计算能力的提升，SHA-1 已被证明存在安全风险，逐步退出历史舞台。由于具备这些特性，Hash 算法非常适合用于数据完整性校验，是数字签名的重要组成部分。

7.1.5.2.数字签名的工作原理（重点★★★★★）

1. 签名的创建过程。对需要签名的文件进行 Hash 计算，得到消息摘要。使用签名者的私钥对消息摘要进行加密，生成数字签名。可以选择加入时间戳，标识签名的创建时间。最终发送的内容包括原始文件和数字签名。需要注意，数字签名只对文件的 Hash 值进行签名，而不是直接对整个文件签名，这样既节约了资源，又提高了效率。

2. 签名的验证过程。接收方对收到的原始文件进行 Hash 计算，得到本地摘要。使用签名者的公钥解密数字签名，得到发送时的摘要。比较两个摘要，如果一致，则文件完整且签名有效；若不一致，则说明文件被篡改或签名无效。

数字签名创建与验证流程示意图



7.1.6. 数字证书（重点★★★★★）

在数字签名的过程中，私钥用于加密消息的摘要，而公钥则用于对端解密以进行认证。这个过程虽然能有效验证信息的完整性和来源，但也引出了一个关键问题：如何确保公钥确实属于消息的发送者，而不是被恶意篡改或伪造的？这个问题正是数字证书和证书认证体系的核心。

当接收方使用发送方的公钥来验证数字签名时，如果公钥在传输过程中被篡改或伪造，接收方便无法正确验证签名，从而导致安全漏洞。为了避免这种情况，就需要确保公钥的真实性——这正是数字证书的作用所在。

数字证书通过引入证书中心（CA）来解决这个问题。CA 充当了一个信任的第三方，它会验证公钥的归属关系，并通过自己的私钥对公钥及其相关信息进行加密，生成数字证书。接收方可以通过 CA 的公钥解密证书，验证公钥的真实性，确保收到的信息来自真实的发送者，而非被恶意篡改。

下面是一个典型的过程。首先，客户端（如浏览器）向服务器发起加密通信请求。服务器响应时，会返回自己的数字证书，并使用私钥对信息进行加密。接着，客户端使用证书中的 CA 公钥解密证书，并验证其有效性，以确保证书由受信任的 CA 签发且服务器身份无误。如果验证通过，客户端就可以使用服务器的公钥加密敏感信息，并与服务器安全地交换数据。这一过程中，数字证书确保了双方身份的真实性和数据的加密保护。

这里引申出非对称公钥的使用场景。这里我们看到的多是信源验证场景，验证信息来自于某个我们可靠的信息源。此时是通过私钥加密信息（摘要）形成数字签名，然后把公钥+信息+数字签名发送出来，在接收端公钥用于解密信息，并对传递过来的信息做一个摘要，然后进行比较，假如匹配说明信源可靠。

但是公钥私钥同样可以用于加密场景（比如交换对称加密的密钥，因为非对称加密效率低，对称加密效率高。但是对称加密双方要约定一个密钥用于通讯，所以此时出现了这个场景）。前面在签名信源验证场景，我们已经通过数字证书的形式把公钥带给客户端。此时客户端可就用此公钥加密双方通信所需密钥，然后再传给服务端。服务端收到加密信息后，就可以用对应的私钥解密，能拿到密钥。

8.上篇-微服务模式（新教材★★★★★）

微服务架构将单一应用拆分成多个微服务，可单独地开发、管理和扩展每个服务。微服务架构优势包括：解耦复杂应用、独立部署、技术选型灵活、容错性好、易扩展，这一章超级重点不看必挂。

8.1.微服务设计原则（论文用得到）

微服务设计原则和 SOA 的设计原则在理论上略有区别，所以这里单独列出来。这一节的

内容基本不会考概念题，至少不会让你默写，不需要死记硬背。

原则名称	原则内容	实现方式
单一职责原则	每个微服务应只关注一个业务领域，提供一个明确功能，避免微服务间耦合	无特定技术手段，从业务功能划分角度设计
隔离性原则	微服务设计为独立进程，拥有自己的数据库和其他资源	使用容器技术，如 Docker 提供隔离环境
自治性原则	每个微服务具有自身的生命周期、状态和行为	将微服务分解为更小独立组件，各组件有自己生命周期和状态，通过消息传递通信
弹性原则	微服务应能快速适应变化的负载和需求	采用自动化扩展和缩放机制，实现容错和故障恢复机制，如负载均衡、故障转移
可观察性原则	微服务可收集分析自身状态和行为数据	运用日志、指标、跟踪机制，记录活动、收集性能数据、跟踪请求路径
可测试性原则	微服务可进行自动化测试和集成测试	利用自动化测试工具和技术，如单元测试、集成测试，结合持续集成和持续部署
可维护性原则	微服务可快速维护和修改	使用标准化 API 和文档，如 RESTful API、OpenAPI，规范命名和版本控制
安全性原则	微服务提供足够保护和安全措施	运用加密、认证和授权技术，如 TLS/SSL 协议、OAuth2、OpenID Connect
智能端点与简单消息传递原则	服务端点智能独立，服务间通信为简单消息传递，避免复杂交互	无特定技术手段，从通信模式设计角度出发

8.2. 微服务设计模式（重点★★★★★）

这一块架构展开的不多，系分教材把微服务模式实际上分为六大类，这六大类非常有可能

给你一个图例让你谈谈它的处理流程，如下表所示。

名称	定义	图示
聚合器微服务	将多个微服务的 API 组合成统一接口，整合响应后返回客户端，简化交互复杂度。	<pre> graph TD LoadBalancer[负载均衡器] --> Aggregator[聚合器] Aggregator <--> ServiceA[服务 A WAR] Aggregator <--> ServiceB[服务 B WAR] Aggregator <--> ServiceC[服务 C WAR] ServiceA <--> Cache[缓存] ServiceA <--> Database[数据库] ServiceB <--> Cache[缓存] ServiceB <--> Database[数据库] ServiceC <--> Cache[缓存] ServiceC <--> Database[数据库] </pre> <p>图 20-1 聚合器微服务模式示意图</p>
代理微服务	在客户端与微服务间增加中间层，处理路由、负载均衡、安全验证等功能。	<pre> graph TD LoadBalancer[负载均衡器] --> Proxy[代理] Proxy <--> ServiceA[服务 A WAR] Proxy <--> ServiceB[服务 B WAR] Proxy <--> ServiceC[服务 C WAR] ServiceA <--> Cache[缓存] ServiceA <--> Database[数据库] ServiceB <--> Cache[缓存] ServiceB <--> Database[数据库] ServiceC <--> Cache[缓存] ServiceC <--> Database[数据库] </pre> <p>图 20-2 代理微服务模式示意图</p>
链式微服务	微服务按顺序串联，每个服务处理部分功能并传递结果，形成完整流程。	<pre> graph TD LoadBalancer[负载均衡器] --> ServiceA[服务 A WAR] ServiceA <--> Cache1[缓存] ServiceA <--> Database1[数据库] ServiceA --> ServiceB[服务 B WAR] ServiceB <--> Cache2[缓存] ServiceB <--> Database2[数据库] ServiceB --> ServiceC[服务 C WAR] ServiceC <--> Cache3[缓存] ServiceC <--> Database3[数据库] </pre> <p>图 20-3 链式微服务模式示意图</p>

分支微服务	<p>分支微服务是链式微服务扩展，将微服务组成不同分支代表业务场景，可链式调用，依条件灵活行动，分支独立，可执行、跳转，串并行依需求而定。</p>	
数据共享微服务	<p>提供统一数据访问接口或复制机制，支持多应用 / 服务共享数据。</p>	
异步消息微服务	<p>通过消息队列异步通信，解耦服务依赖，提升系统吞吐量和响应能力。</p>	

8.3. 微服务开发（重点★★★★★）

微服务系统开发需要考虑服务交互与数据共享，保障可用性、可靠性、安全性。要对应用分析拆分，明确微服务功能、接口及依赖，进行单元、集成、端到端测试，并用自动化部署和

容器化技术进行部署扩展。这里有很多技术选型的对比，需要重点关注。其中容器已经在云原生架构提到过，所以不再展开。

8.3.1.服务注册和发现简介（次重点★★★☆☆）

服务注册与发现的基本原理是：服务提供者将自己的服务注册到注册中心，服务消费者从注册中心中获取服务的相关信息，然后通过该信息来调用服务提供者。这样就实现了服务提供者和服务消费者之间的解耦，同时也可以保证服务的高可用性和扩展性。常用的技术有 Consul、ZooKeeper、Eureka、Etcd，选词填空的时候要看得出来。

技术名称	核心功能与原理
Consul	服务注册到中心注册表，客户端通过 API 发现服务，可健康检查、支持多数据中心与 KV 存储
ZooKeeper	以集群方式运行，用 ZAB 协议（ZAB 协议即 ZooKeeper 原子广播协议，是为分布式协调服务设计的一种原子广播协议）保证数据一致，提供服务注册与发现及分布式协调功能
Eureka	Server 接收注册、维护清单，Client 注册自身并获取信息，通过心跳维持服务状态
Etcd	基于 Raft 协议存储键值对数据，支持多种操作与监听机制，用作服务注册与配置中心

8.3.2.服务通信（超级重点★★★★★）

在微服务通信技术中，主要有同步调用、异步调用两种模式。同步调用是指客户端调用服务提供者时，必须等待服务提供者响应后才能继续执行。同步调用的优点是简单直接、易于实现和调试，缺点是容易出现阻塞和延迟。异步调用是指客户端调用服务提供者时，不需要等待服务提供者响应，而是先返回一个异步处理的结果，等到服务提供者响应后再回调通知客户端。

异步调用的优点是能够提高响应速度和吞吐量，缺点是实现和调试较为复杂。这一章涉及到多种技术选型的对比，很可能出概念题，主要是通信协议上的对比。

通信协议	核心要点	在微服务架构中的优势
HTTP /REST	HTTP 是用于网络数据通信的协议，REST 是基于 HTTP 的软件架构风格。	简单易用，降低系统复杂度；具有语言无关性，方便实现跨语言通信；易于扩展，可自定义头部。
gRPC	高性能、开源的 RPC 框架，基于 HTTP/2 协议设计，使用 ProtoBuf 作为 IDL 定义服务接口和消息格式。	基于 HTTP/2 具备多路复用等特性，传输效率高，且支持 TLS/SSL 加密保障安全；实现服务间高效通信，降低耦合度。

8.3.2.1.gRPC (超级重点★★★★★)

gRPC 是一个高性能、开源的远程过程调用（Remote Procedure Call，RPC）框架，它包含四种不同的通信模式，需要看到名字知道它们对应的通信方式。

RPC 通信模式	模式特点
一元 RPC	单一请求和响应模式，与传统的 HTTP 请求和响应模式类似
服务器流 RPC	客户端发出一次请求，服务器端进行多次响应
客户端流 RPC	客户端多次发送请求，服务器端进行一次响应
双向流 RPC	双向流模式，能在同一个连接上实现客户端和服务器端的双向流式通信

8.3.2.2.Kafka (超级重点★★★★★)

在异步调用模式下，会需要用到消息队列。这里书上提到了 Kafka。Kafka 是一种分布式流事件处理平台，最初由 LinkedIn 开发，现在是 Apache 基金会的一部分。它的核心功能主要包括消息队列、流处理和数据集成，其关键技术特点如下。

特点	具体内容	简单介绍
分区和副本策略	每个主题可分成多个分区，每个分区有多个副本	在 Kafka 架构中，主题是消息的逻辑分类。通过将主题拆分为多个分区，实现数据并行处理，提升整体处理能力。每个分区的多个副本分布在不同节点，当某个节点故障时，其他副本所在节点能继续提供服务，防止数据丢失，确保数据高可用与持久存储，满足大数据量并发读写需求
高效 I/O 模型	数据写入操作采用顺序追加，避免磁盘随机读写	传统磁盘随机读写时，磁头需频繁移动寻址，耗时较长。Kafka 采用顺序追加写入方式，磁头只需顺序移动，无需频繁寻址，大大减少了磁盘 I/O 操作的时间开销，显著提升数据写入磁盘的速度，进而提高整体写入性能
零拷贝技术	采用零拷贝技术提升数据传输效率	零拷贝技术指数据在传输过程中，无需经过用户态内存的拷贝，直接在内核态完成数据从磁盘到网络的传输。减少了数据拷贝次数，降低 CPU 开销，加快数据传输速度，有效降低数据传输延迟，使 Kafka 在数据传输环节更加高效

8.3.3.SpringCloud 微服务框架图 (重点★★★★★)

下图是系分教材中出现的 SpringCloud 架构图，这里涉及到的技术，比如什么是 Feign、eureka、ribbon 你要知道是什么，需要梳理出来记忆。

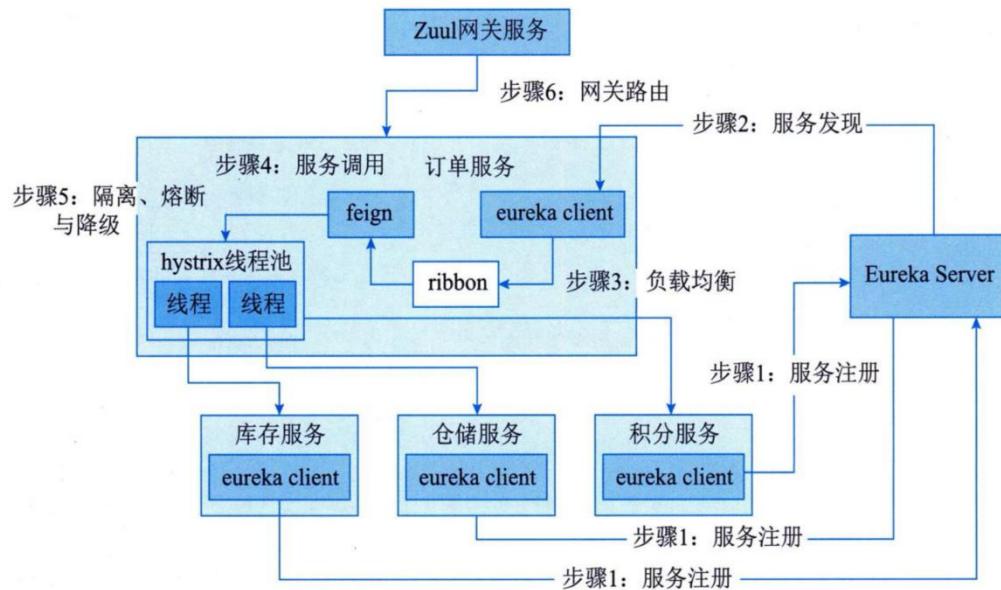


图 18-9 SpringCloud 的微服务示意图

服务注册与发现。Eureka Server：作为服务注册中心，是 Spring Cloud Netflix 组件之一。各微服务(如库存服务、仓储服务、积分服务、订单服务)的 eureka client 会将自身注册到 Eureka Server，实现服务注册（步骤 1）。其他服务可通过 Eureka Server 发现所需服务（步骤 2），解决微服务间的地址管理和发现问题。

负载均衡。Ribbon：是客户端负载均衡器，属于 Spring Cloud Netflix 组件之一。与 eureka client 配合，在订单服务调用其他服务（如库存服务等）时，从 Eureka Server 获取到多个服务实例地址后，按特定算法（如轮询、随机等）选择一个实例进行调用，实现负载均衡（步骤 3）。

服务调用。Feign：是声明式的 Web 服务客户端，基于 Ribbon 和 Hystrix。让开发者能以声明接口的方式编写服务调用代码，像调用本地方法一样调用其他微服务，简化服务调用过程（步骤 4）。

服务容错。Hystrix：用于实现服务的隔离、熔断与降级（步骤 5）。通过 hystrix 线程池为每个依赖服务分配独立线程资源，隔离故障；当依赖服务调用失败率等指标达到阈值，触发熔断，不再调用故障服务；还能进行服务降级，在故障时返回兜底数据或提示信息，保障系统

稳定性。

网关。Zuul：是网关服务，作为系统的统一入口。接收外部请求，根据配置进行路由转发（步骤 6），还可实现请求过滤、鉴权、限流等功能，保护内部微服务并进行流量管控。

8.4. 安全与权限管理（重点★★★★★）

这一块凯恩认为重点关注 OAuth 2.0 和 JWT，主要是它们的结构和工作流程，案例会考。

8.4.1. OAuth 2.0（重点★★★★★）

OAuth 2.0 是一种授权框架，旨在为用户提供一种安全且便捷的方式，让第三方应用能够在有限的范围内访问用户在另一个服务提供商（如社交媒体平台、云存储等）上的资源，而无需用户直接将自己的用户名和密码提供给第三方应用。

四个主要角色：客户端：即第三方应用，它请求访问受保护的资源。资源所有者：通常是指用户，拥有受保护的资源，并决定是否授权客户端访问这些资源。授权服务器：负责验证资源所有者的身份，并在资源所有者授权后，向客户端颁发访问令牌。资源服务器：托管受保护资源的服务器，它根据访问令牌来验证客户端的身份，并决定是否允许客户端访问相应的资源。

四种授权模式：（1）授权码模式。用于 Web 应用程序的授权流程，将授权码作为临时访问令牌，用于获取访问令牌。（2）密码模式。适用于受信任的客户端（如本地应用程序），客户端直接通过用户凭据向授权服务器申请访问令牌。（3）客户端模式。适用于客户端自身访问资源，客户端直接通过客户端凭据向授权服务器申请访问令牌。（4）**隐式授权模式（最重要）**。适用于移动应用程序或 Web 应用程序，将访问令牌直接从授权服务器发送到客户端，而不需要授权码。（2 和 4 的区别在于输入密码位置不同，2 在第三方客户端你肯定不放心，4 是在授权服务器验证输入）

拓展阅读（四个角色+隐式授权模式）

客户端：假设存在一个名为“社交聚合器”的第三方应用。该应用旨在整合多个社交平台的信息，为用户提供一个统一的社交信息浏览和管理界面。它希望获取用户在微博上的信息，如微博内容、关注列表等，以展示给用户并进行相关的分析和整合操作。

资源所有者：普通微博用户小明是资源所有者。小明在微博上发布了许多微博，拥有自己的关注列表、粉丝列表等信息，这些都是受保护的资源。小明希望在“社交聚合器”应用中能够方便地查看自己在微博以及其他社交平台上的综合信息，所以需要决定是否授权“社交聚合器”访问其微博资源。

授权服务器：微博的授权服务器负责验证用户的身份以及处理授权请求。当小明在“社交聚合器”应用中点击授权登录微博时，“社交聚合器”会将小明重定向到微博的授权服务器页面。小明在该页面输入自己的微博账号和密码进行身份验证，然后选择是否授权“社交聚合器”访问自己的特定微博资源。如果小明授权，授权服务器会生成一个访问令牌，并将其返回给“社交聚合器”应用。

资源服务器：微博的服务器充当资源服务器，存储着用户的各种微博资源，如微博内容、用户资料、关注关系等。当“社交聚合器”应用拿到访问令牌后，会使用该令牌向微博的资源服务器发送请求，获取小明的相关微博资源。资源服务器会验证访问令牌的有效性，如果有效，就会根据令牌所包含的权限信息，向“社交聚合器”应用返回相应的资源数据，例如小明的最近几条微博内容或者他的关注列表等。

8.4.2.JWT（重点★★★★★）

JWT (JSON Web Token) 是一种开放标准 (RFC 7519)，用于在各方之间安全地传输声明。它是一个紧凑且自包含的 JSON 对象，通常由三部分组成，即头部 (Header)、载荷 (Payload)

和签名（Signature）。

工作流程： (1) 用户认证：用户向服务器发送登录请求，服务器验证用户的身份。(2) 生成 JWT：如果用户身份验证成功，服务器将创建一个 JWT，并将其返回给客户端。(3) 客户端存储和使用：客户端接收到 JWT 后，通常会将其存储在本地(如 localStorage 或 cookie 中)。在后续的请求中，客户端会将 JWT 包含在请求头中发送给服务器。(4) 服务器验证：服务器接收到请求后，会验证 JWT 的签名和有效性。如果验证通过，服务器将处理请求并返回响应。

8.4.3.API 网关（非重点☆☆☆☆☆）

API 网关是微服务架构中 API 的入口服务器，介于客户端和后端微服务间。其主要功能有请求路由、过滤、协议转换、流量管理以及监控与日志记录。特点是提供统一入口、安全性高和可扩展性强。应用于微服务架构、多端接入场景以及 API 管理等方面。

9.上篇-面向服务架构（超级重点★★★★★）

面向服务架构（SOA），无论是架构还是系分都是超级重点的存在，属于不看必挂的内容。隔三差五出一道大题，有的是简答对比，有的是选词填空，出题形式特别多样。这一章最基本的还是要掌握 SOA、微服务和单体架构之间的对比。另外微服务/SOA 本身也会是论文的常考点，比如让你从具体实现的角度，让你从设计原则角度展开等，这些也需要关注，虽然不太可能是案例的考察内容，但是凯恩也把它总结出来。

9.1.什么是 SOA（重点★★★★★）

有个印象即可，应付概念简答题。

从应用的角度定义，可以认为 SOA 是一种应用框架，它着眼于日常的业务应用，并将它们划分为单独的业务功能和流程，即所谓的服务。SOA 使用户可以构建、部署和整合这些服务，且无需依赖应用程序及其运行平台，从而提高业务流程的灵活性。这种业务灵活性可使企业加快发展速度，降低总体拥有成本，改善对及时、准确信息的访问。SOA 有助于实现更多的资产重用、更轻松的管理和更快地开发与部署。

从软件的基本原理定义角度，认为 SOA 是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以一种统一和通用的方式进行交互。

9.2.SOA/微服务/单体架构对比（超级重点★★★★★）

这是书本上新增的内容，最近没有考过，值得重点关注。最近考过微服务和单体架构的区别，那么 SOA 和微服务，SOA 和单体就是大概率考的方向。下表是微服务和 SOA 的区别，这里分为 5 个维度，应付考试没问题了。

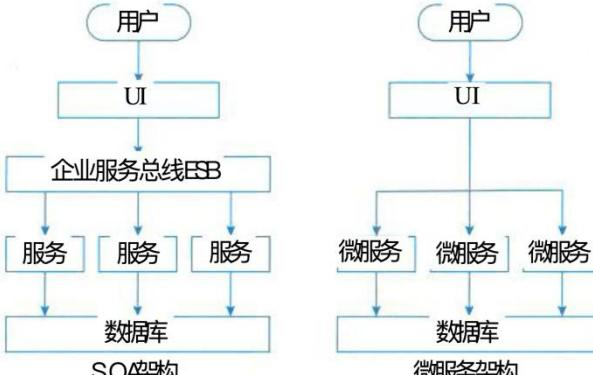
9.2.1.微服务和单体架构对比（超级重点★★★★★）

特性	单体式系统	微服务系统
架构粒度	一个整体的应用程序，所有功能和模块都集中在一起	多个小而自治的服务，每个服务负责特定的业务功能
部署和扩展粒度	整体部署和扩展	每个服务独立部署和扩展
技术异构性	不支持	支持
数据管理	共享一个数据库	每个服务有自己的数据库

团队组织和协作	适合一个开发团队共同开发和维护	适合多团队协作,每个团队独立负责一个或多个服务
可靠性和容错性	不支持	支持
开发难度	简单	复杂

9.2.2.微服务和 SOA 对比 (超级重点★★★★★)

特性	微服务	SOA (面向服务的架构)
服务粒度	服务粒度极为精细, 每个微服务通常仅负责单一的业务功能, 例如在电商系统中, 商品展示、订单处理、支付等功能可分别由独立的微服务实现。这种细粒度的划分使得每个服务专注于特定任务, 便于开发、维护和独立扩展。	服务粒度相对较大, 一个服务可能涵盖多个业务功能。比如在企业资源规划 (ERP) 系统中, 一个服务可能同时包含采购管理和库存管理的相关功能, 其整合程度更高。
通信方式	主要使用 HTTP RESTful API 进行通信, 这种方式具有良好的语言和平台无关性, 不同语言开发的微服务之间可以轻松交互, 并且 RESTful 风格简单易用, 便于与各种客户端进行集成。	通信方式更为多样化, 除了 HTTP 外, 还常使用 SOAP (简单对象访问协议) 等协议。SOAP 具有严格的消息格式和标准, 适用于对数据交互规范性和安全性要求较高的企业级应用场景。
数据管理	每个微服务通常拥有自己独立的数据库, 形成数据源唯一的特性, 确保数据的独立性和安全性。不同微服务的数据库可以根据自身业务需求选择合适的数据库类型, 如关系型数据库、非关系型数据库等。	数据管理方式较为灵活, 可能采用多个服务共享同一个数据库的方式, 或者通过企业服务总线 (ESB) 进行数据交换和整合, 实现不同服务间的数据交互和协调。
架构风格	遵循去中心化的架构风格, 不存在中心化的服务总线。各个微服务独立部署、运行和管理, 通过轻量级的通信机制相互协作, 具有较高的灵活性和可扩展性。	可能包含中心化的企业服务总线 (ESB), ESB 充当服务间通信和数据交换的枢纽, 负责处理服务的注册、发现、路由以及消息转换等功能, 在一定程度上增强了服务间的集成和管理能力。

复杂性	<p>由于服务的独立性和去中心化特性，微服务架构的系统复杂性相对较低。每个微服务可以独立开发、测试和部署，降低了整体系统的耦合度。但同时，微服务数量众多也会带来服务治理、监控和运维等方面的挑战。</p>	<p>由于服务间可能存在较强的耦合关系，并且中心化的 ESB 增加了架构的复杂性，使得 SOA 架构的系统在设计、开发和维护上相对复杂，对团队的技术能力和管理水平要求较高。</p>
 <p style="text-align: center;">图15-1 SOA架构与微服务架构图</p>		

9.2.3.SOA 和单体架构对比 (超级重点★★★★★)

特性	SOA	单体架构
服务粒度	将业务功能分解为多个相对独立、粒度较细的服务，每个服务专注于特定的业务功能，可被不同应用复用，提高了业务灵活性和可扩展性。	整个应用是一个紧密耦合的整体，所有功能模块都集成在一起，难以单独拆分和复用，新增或修改功能可能影响整体。
通信方式	采用多种通信协议，如 SOAP、HTTP 等，服务之间通过标准的接口进行通信，实现不同服务间的交互和数据共享，支持分布式部署。	内部通过方法调用的方式实现模块间交互，无需复杂的外部通信协议，主要在同一应用内部完成。
数据管理	数据管理较为灵活，可以是多个服务共享数据库，也可通过企业服务总线（ESB）进行数据交换和整合，实现不同服务间的数据交互和协调。	通常使用单一数据库，所有功能模块都直接访问和操作同一数据源，数据的一致性和管理相对集中。
架构风格	属于分布式架构，各个服务可以独立部署和运行，可能存在中心化的 ESB，用于协调服务间的通信和数据交换，实现服务的集成和管理。	是集中式架构，所有功能集中在一个应用程序中，运行在同一环境下，易于开发和部署，但后期维护和扩展较困难。

故障影响	某个服务出现故障时，一般不会对其他服务产生直接影响，仅会影响依赖该服务的特定功能，故障隔离性较好。	一旦应用中的某个部分出现故障，可能会导致整个应用无法正常运行，故障影响范围大。
------	---	---

9.3.服务注册模式（重点★★★★★）

服务注册模式和 ESB 模式书本上是当作两种不同的模式分开介绍的，在真题中都考察过。

9.3.1.服务注册模式概念（重点★★★★★）

服务注册模式包含服务提供者、服务请求者和服务注册，它们的定义如下表所示。

角色	操作内容
服务提供者	使用 UDDI 将 Web 服务发布到服务注册中心，通过 WSDL 描述服务
服务注册中心	接收并存储服务提供者通过 UDDI 发布的服务信息，供服务请求者查找
服务请求者	借助 UDDI 在服务注册中心查找 Web 服务，依据 WSDL 了解调用方式，通过 SOAP 与服务提供者建立连接并使用服务

9.3.2.UDDI、WSDL 和 SOAP（重点★★★★★）

服务注册模式包含服务提供者、服务请求者和服务注册，它们之间通过 UDDI、WSDL 和 SOAP 三种协议协同工作。这些协议的概念、作用是必须要掌握的。如下表所示。

协议名称	概念	主要作用
UDDI（通用描述、发现和集成服务）	是一个用于发现和集成 Web 服务的协议标准，基于 XML 的跨平台的描述规范	提供统一的方式发布和搜索 Web 服务信息。服务提供者可在注册表注册服务，供使用者查找和连接

WSDL (Web 服务描述语言)	用于描述 Web 服务接口的 XML 格式语言	定义 Web 服务的功能、输入输出参数、绑定协议和访问端点等信息，让服务使用者了解如何调用和使用 Web 服务
SOAP (简单对象访问协议)	基于 XML 的消息传输协议，用于在 Web 服务之间进行通信	定义消息格式、编码规则和 RPC 约定，用于在应用程序之间交换结构化信息，可与多种因特网协议和格式结合使用，如 HTTP、SMTP 等

9.3.3.REST 规范/风格 (重点★★★★★)

这一块考的次数比较频繁，需要好好关注。书本上写的概念有点简略，不足以应付考试。
凯恩在这里补充如下。

REST 全称是 Representational State Transfer，中文意思是表述性状态转移。它首次出现在 2000 年 Roy Fielding 的博士论文中。如果一个架构符合 REST 的约束条件和原则，我们就称它为 RESTful 架构。REST 本身并没有创造新的技术、组件或服务，而是更好地使用现有 Web 标准中的一些准则和约束，如下所示。

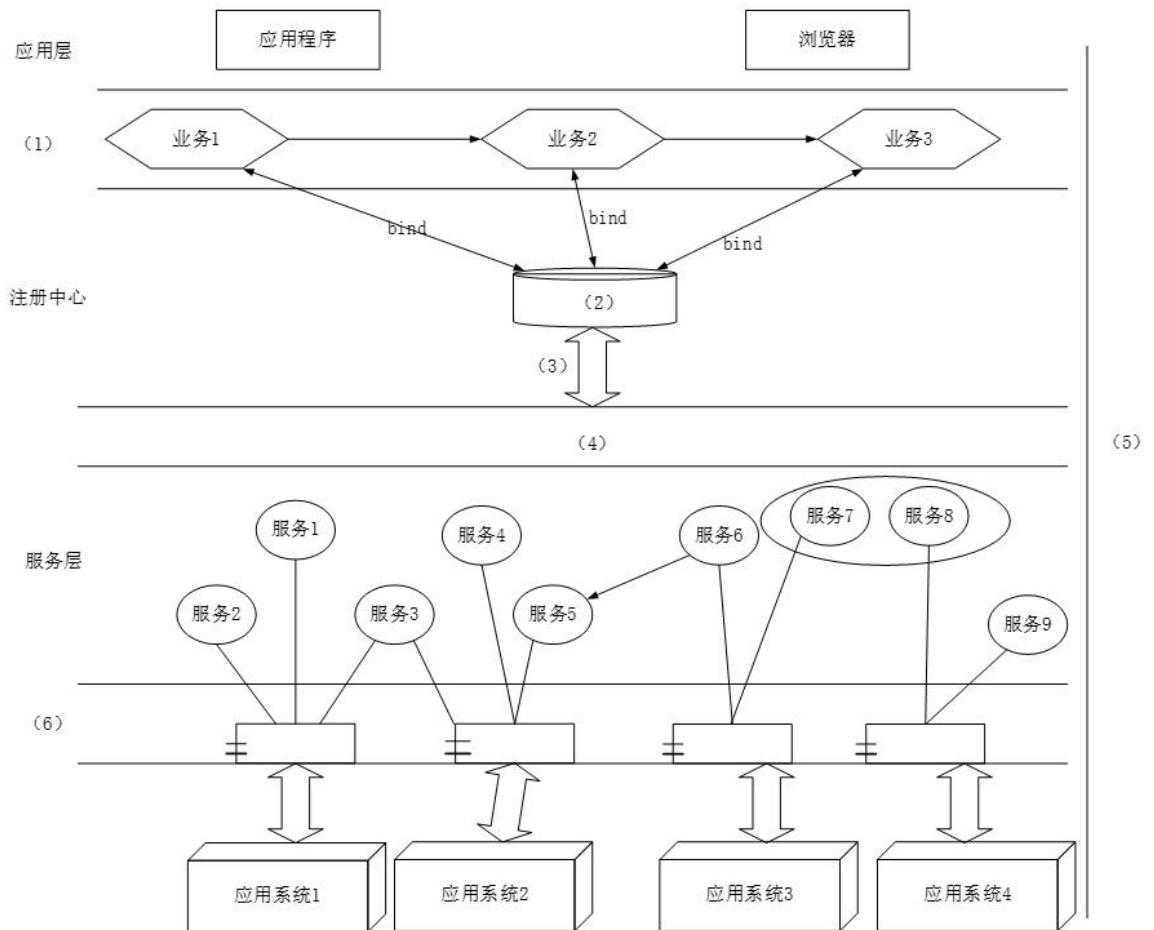
主题	内容要点
资源与 URI	资源是任何有被引用必要的事物，包括实体和抽象概念。URI 是资源唯一标识，可看作资源地址或名称。
统一资源接口	遵循统一接口原则，使用标准 HTTP 方法 (GET、PUT、POST、DELETE 等) 访问资源。

资源的表述	客户端获取的是资源的表述，资源表述形式多样，如文本可采用 html、xml、json 等格式。
资源的链接	REST 需利用超媒体概念，在表述格式中加入链接引导客户端，增强资源连通性。
状态的转移	服务端不保存客户端状态。客户端与服务端交互无状态，每次请求包含所需信息，从而可推进状态转移。

9.3.4.典型例题（重点★★★★★）

(18 年架构真题) 某银行拟将以分行为主体的银行信息系统，全面整合为由总行统一管理维护的银行信息系统，实现统一的用户账户管理、转账汇款、自助缴费、理财投资、贷款管理、网上支付、财务报表分析等业务功能。但是，由于原有以分行为主体的银行信息系统中，多个业务系统采用异构平台、数据库和中间件，使用的报文交换标准和通信协议也不尽相同，使用传统的 EAI 解决方案根本无法实现新的业务模式下异构系统间灵活的交互和集成。因此，为了以最小的系统改进整合现有的基于不同技术实现的银行业务系统，该银行拟采用基于 ESB 的面向服务架构 (SOA) 集成方案实现业务整合。

问：基于该信息系统整合的实际需求，项目组完成了基于 SOA 的银行信息系统架构设计方案。该系统架构图如图所示：



请从 (a) ~ (j) 中选择相应内容填入上图 (1) ~ (6) 处，补充完善架构设计图。

- (a) 数据层 (b) 界面层 (c) 业务层 (d) bind (e) 企业服务总线 ESB (f) XML
 (g) 安全验证和质量管理 (h) publish (i) UDDI (j) 组件层 (k) BPEL

- (1) 填 (c) 业务层：图中该位置处于应用层下方，包含业务 1、业务 2、业务 3，主要负责处理具体业务逻辑，符合业务层的定义，所以是业务层。
- (2) 填 (i) UDDI：这一行左侧写着注册中心，已经给出提示，注册中心这里的主要功能是服务的注册与发现，UDDI（统一描述、发现和集成协议）就是专门用于实现这一功能的，业务通过“bind”操作与它交互来查找服务，因此此处为 UDDI。
- (3) 填 (h) publish：服务要能够被业务发现和调用，需要先在注册中心进行发布操作，publish就是发布的意思，所以该位置应填 publish。

- (4) 填 (e) 企业服务总线 ESB：从架构层次关系看，该位置处于注册中心和服务层之间，起到连接和通信的作用，企业服务总线 ESB 能实现不同服务之间的消息传递、协议转换等，符合此层的功能需求，故为 ESB。这个图也说明 ESB 和服务注册模式可以混用。
- (5) 填 (g) 安全验证和质量管理：在系统架构的各个层次中，都需要对服务的安全性和质量进行管控，负责对服务调用进行安全验证以及质量管理，所以是安全验证和质量管理。
- (6) 填 (j) 组件层：此层位于服务和应用系统之间，其作用是将应用系统的功能封装成可复用的服务组件，供外部系统调用，所以是组件层。

(1) (c) 业务层 (2) (i) UDDI (3) (h) publish (4) (e) 企业服务总线 ESB (5) (g) 安全验证和质量管理 (6) (j) 组件层 (实际考试你写英文序号就行不用写后面的代表的具体的内容)

9.4.ESB 模式 (重点★★★★★)

9.4.1.ESB 概念 (重点★★★★★)

书本上把 ESB 当作是另外一种单独的，和之前提到的服务注册表模式有区别的设计模式。在引入 ESB 之前，书本花了不少篇幅来讲什么是 IBM Websphere。但是凯恩调研了一下，这个技术底座虽然已经有二十多年的历史了，但是，相关资料奇少无比，应该不是重点。根据真题经验，这里只要掌握 ESB (书本定义为架构模式) 的相关概念即可。

企业服务总线 (ESB) 模式由消息中间件发展而来，采用“总线”模式管理简化应用集成拓扑，基于开放标准支持应用在消息、事件和服务级别的互联互通。ESB 提供位置透明性的消息路由和寻址服务、服务注册和命名的管理功能，支持多种消息传递范型（如请求 / 响应、发布 / 订阅等）、多种可以广泛使用的传输协议、多种数据格式及其相互转换，还提供日志和监控功能。

9.4.2. 典型例题（重点★★★★★）

(20 年架构真题) 某银行拟将以分行为主体的银行信息系统，全面整合为由总行统一管理维护的银行信息系统，实现统一的用户账户管理、转账汇款、自助缴费、理财投资、贷款管理、网上支付、财务报表分析等业务功能。但是，由于原有以分行为主体的银行信息系统中，多个业务系统采用异构平台、数据库和中间件，使用的报文交换标准和通信协议也不尽相同，使用传统的 EAI 解决方案根本无法实现新的业务模式下异构系统间灵活的交互和集成。因此，为了以最小的系统改进整合现有的基于不同技术实现的银行业务系统，该银行拟采用基于 ESB 的面向服务架构 ((SOA) 集成方案实现业务整合。

问：请分别用 200 字以内的文字说明什么是面向服务架构 (SOA) 以及 ESB 在 SOA 中的作用与特点。

此题直接考察 SOA 的概念和 ESB 的概念，直接默写。2018 年架构也考过一模一样的问题。

答：SOA 是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以一种统一和通用的方式进行交互。

企业服务总线 (ESB) 提供位置透明性的消息路由和寻址服务、服务注册和命名的管理功能，支持多种消息传递范型（如请求 / 响应、发布 / 订阅等）、多种可以广泛使用的传输协议、多种数据格式及其相互转换，还提供日志和监控功能。

9.5. SOA 设计原则（重点★★★★★）

SOA 的设计需要遵循哪些原则，真题没有考过，下面的内容可以用口诀加以记忆。我们可以提取每个要点的首字，组成 “无单明自粗松重互” —— 想象没有订单的人，自己的名字

就会变得粗劣松散，失去重要客户。

特性	简要概括
无状态	避免服务请求者依赖服务提供者状态
单一实例	防止功能冗余
明确定义的接口	由 WSDL 定义，其中 WS - Policy 描述规约，XML 模式定义消息格式等
自包含和模块化	封装组件，实现功能实体独立部署、管理
粗粒度	服务数量适度，靠消息交互
松耦合性	使用者仅见接口，服务位置、实现等对其不可见
重用能力	服务具备可重用性
互操作性、兼容和策略声明	确保不同的系统、应用程序、设备或组件之间能够相互通信

10. 上篇-系统规划（重点★★★★★）

系统规划这一章节出过案例题，特别是成本效益的计算还有可行性研究都出过，属于要么不出一出就一堆人翻车的内容，有余力的同学还是要硬啃这里的计算。

10.1. 项目立项方法（次重点★★★☆☆）

可能案例出概念，大致了解步骤就行。都是字面意思。用首字记忆法“选评优评平”记忆。

(1) 选择有核心价值的项目。(2) 评估所选择的项目的约束、风险、成本和效益。(3) 项目优先级排序。(4) 评估项目的多种实施方式(外包、自研、购买服务)。(5) 平衡各类风险(技术风险，目标风险，继承风险等)。

10.2. 可行性研究（重点★★★★★）

10.2.1. 可行性研究的四个方面（重点★★★★★）

在信息系统建设项目中，可行性研究通常从经济可行性、技术可行性、法律可行性和用户使用可行性（必须记住）4 个方面来进行分析，其中经济可行性通常被认为是项目的底线。

可行性研究	定义
经济可行性	从成本效益角度分析项目在经济上是否合理，包括项目的建设成本、运营成本、收益等方面 的评估，判断项目是否能带来经济效益，通常被视为项目的底线。
技术可行性	评估项目所涉及的技术是否成熟、可靠，现有技术水平能否实现项目的功能和性能要求，以及技术的发展趋势对项目的影响等。
法律可行性	分析项目是否符合法律法规、政策要求，是否存在知识产权、版权、隐私保护等方面的法律问题。
用户使用可行性	从用户角度出发，评估项目功能和特性是否符合用户的业务需求和使用习惯，用户对系统的接受程度和操作便捷性等。

10.2.2. 典型例题（重点★★★★★）

(16 年系分) 某软件开发企业受对外贸易公司委托开发一套跨境电子商务系统，项目组从多个方面对该电子商务系统进行了可行性分析，在项目组给出的可行性分析报告中，对项目的成本、收益情况进行了说明：建设投资总额为 300 万元，建设期为 1 年，运营期为 4 年。

问：软件系统可行性分析包括哪几个方面？用 200 以内文字说明其含义。

软件系统可行性分析的内容和基本概念是进行软件系统可行性分析的前提。软件系统的可行性分析包括经济可行性、技术可行性、法律可行性和用户使用可行性，这个题目只要对概念进行解释即可。

答：经济可行性。主要评估项目的建设成本、运行成本和项目建成后可能的经济收益。技

术可行性。研究的对象是信息系统需要实现的功能和性能，以及技术能力约束。**法律可行性**。具有比较广泛的内容，它需要从政策、法律、道德、制度等社会因素来论证信息系统建设的现实性。**用户使用可行性**。从用户角度出发，评估项目的功能和特性是否符合用户的业务需求和使用习惯，用户对系统的接受程度和操作便捷性等。

10.3.成本和收益（次重点★★★☆☆）

选择题案例题会给你一个描述让你判断属于那种成本，所以也挺关键。

10.3.1.概念解释（次重点★★★☆☆）

成本按照投资时间分为，为基础建设投资、其他一次性投资和非一次性投资三大类。按照成本性态分类，可以分为固定成本(管理人员的工资、办公费、固定资产折旧费、员工培训费)、变动成本(直接材料费、产品包装费、外包费用、开发奖金)和混合成本(例如，水电费、电话费)。软件生存期中的成本按照系统投入的阶段又可分为开发阶段的成本和建成后的运营成本两类(真题中会体现)。

收益可以分为有形收益和无形收益。有形收益也称为经济收益，可以用货币的时间价值、投资回收期、投资回收率等指标进行度量。无形收益也称为不可定量的收益，一般是企业形象提升、员工满意度提高、决策效率提升等等。

10.3.2.典型例题（重点★★★★★）

(16年系分)某软件开发企业受对外贸易公司委托开发一套跨境电子商务系统，项目组从多个方面对该电子商务系统进行了可行性分析，在项目组给出的可行性分析报告中，对项目的成本、收益情况进行了说明：建设投资总额为300万元，建设期为1年，运营期为4年。

问：成本和收益是经济可行性评价的核心要素，成本一般分为开发成本和运营成本，收益包括有形收益和无形收益，请对照下列 7 项内容，将其序号分别填入成本和收益对应的类别。

系统分析师工资； b) 采购数据库服务器； c) 系统管理员工资； d) 客户满意度增加；
e) 销售额同比提高； f) 软件许可证费用； g) 应用服务器数量减少。

类别	选项
开发成本	(1)
运营成本	(2)
有形收益	(3)
无形收益	(4)

成本和收益的识别是经济可行性中成本效益分析的基础，成本主要描述系统的投入，收益主要描述系统建成后的产出。软件生存期中的成本按照系统投入的阶段可分为开发阶段的成本和建成后的运营成本两类，软件生存期中的收益按照量化放肆不同可分为有形收益和无形收益。系统分析师工资和采购数据库服务器属于系统开发阶段的投入，所以属于开发成本；系统管理员工资和软件许可证费用属于系统建成后运营阶段的投入，所以属于运营成本；销售额同比提高和应用服务器数量减少都意味着系统收益的增加，可以直接量化，所以属于有形收益；而客户满意度增加同样能够增加系统收益，但是无法直接量化，所以属于无形收益。

答：(1) 开发成本：(a) 系统分析师工资、(b) 采购数据库服务器 (2) 运营成本：(c) 系统管理员工资、(f) 软件许可证费用 (3) 有形收益：(e) 销售额同比提高、(g) 应用服务器数量减少 (4) 无形收益：(d) 客户满意度增加

10.4. 成本效益计算（重点★★★★★）

10.4.1. 公式汇总（重点★★★★★）

公式名称	公式内容	备注

净现值	$P = \frac{F}{(1+i)^n}$	项目在生命周期内各年净现金流量按折现率折现到初始时的现值之和
静态投资回收	累计净现金流量开始出现正值的年份数 $- 1 + \frac{ \text{上年累计净现金流量} }{\text{当年净现金流量}}$	不考虑货币时间价值，从项目开始投入之日起，包括建设期
动态投资回收期	累计折现值开始出现正值的年份数 $- 1 + \frac{ \text{上年累计折现值} }{\text{当年折现值}}$	考虑货币时间价值，从项目开始投入之日起，包括建设期
净现值率	净现值率就是净现值和投资净现值的比值	净现值与投资净现值的比值
投资收益率	投资收益率=投资收益/投资成本×100%	反映项目投资盈利水平
投资回收率	投资回收率=1/动态投资回收期 × 100%	反映项目投资回收能力

10.4.2. 净现值（重点★★★★★）

所谓的净值就是未来的钱通过逆向的复利公式折算到现在。若 n 年后能收入 F 元，那么这些钱现在的价值（通常简称为“现值”）P 是，这里的 i 就是银行利率。

$$P = \frac{F}{(1+i)^n}$$

净现值是指项目在生命周期内各年的净现金流量（净现金流量是收入-成本支出）按照一定的、相同的折现率折现到初时的现值之和。如下公式所示。

$$NPV = \sum_{t=0}^n \frac{(CI - CO)_t}{(1+i)^t}$$

其中 (CI-CO) 为第 t 年的净现金流量，CI 为现金流入，CO 为现金流出，i 为折现率。

10.4.3. 净现值率（重点★★★★★）

净现值率就是净现值和投资净现值的比值。

10.4.4. 投资回收期与投资回报率（重点★★★★★）

计算投资回收期时，根据是否考虑货币是否贬值，可分为静态投资回收期和动态投资回收期（考虑货币贬值）。投资回收期从信息系统项目开始投入之日算起，即包括建设期，单位通常用“年”表示，这里会有一个隐藏的细节。这里凯恩还是以上面的例子为例来进行解释。

10.4.4.1. 静态投资回收期（重点★★★★★）

$$T = \text{累计净现金流量开始出现正值的年份数} - 1 + \frac{|\text{上年累计净现金流量}|}{\text{当年净现金流量}}。$$

表 10-7 各年度的折现系数和折现值 (单位: 万元)

方案	建设期			运营期				
	0	1	合计	2	3	4	5	合计
折现系数	1	0.91		0.83	0.75	0.68	0.62	
甲	年初投资额	350	150	500				
	年末净现金流量			150	200	250	400	1000
	折现值	350	136.5	486.5	124.5	150	170	248
乙	年初投资额	300	200	500				
	年末净现金流量			100	200	300	400	1000
	折现值	300	182	482	83	150	204	248
丙	年初投资额	400	100	500				
	年末净现金流量			200	250	250	300	1000
	折现值	400	91	491	166	187.5	170	186

那么从上表中我们知道

$$(1) \text{ 甲方案的静态投资回收期为: } (4 - 1) + \frac{|-150|}{250} = 3.6 \text{ 年}$$

$$(2) \text{ 乙方案的静态投资回收期为: } (4 - 1) + \frac{|-200|}{300} = 3.67 \text{ 年}$$

$$(3) \text{ 丙方案的静态投资回收期为: } (4 - 1) + \frac{|-50|}{250} = 3.2 \text{ 年}$$

这里看看甲方案，计算累计净现金流量：第 0 年：-350；第 1 年：-350-150=-500；第 2 年：-500+150=-350；第 3 年：-350+200=-150；第 4 年：-150+250=100（首次出现正值）。

代入公式，甲方案的静态投资回收期为： $(4 - 1) + \frac{|-150|}{250} = 3.6 \text{ 年}$ 。

这里的计算结果好像不符合常识，因为仔细看题目，我们建设期用了 2 年，运营期用了 2.6 年，那应该是 4.6 年才对。这里你要注意表格里的年份都是从 0 开始计算的，我们代入的年份也是从 0 开始计算的，所以有 -1 的偏差，也就是我们的 3.6 不是绝对意义上的花了 3.6 年，而是说在表格里的这个 3.6 年的时间点（从 0 开始标记）我们完成回收。这个不需要深究，一律代入公式就对了。其他方案以此类推可以自行计算。

10.4.4.2. 动态投资回收期与投资回报率（重点★★★★★）

$$T = \text{累计折现值开始出现正值的年份数} - 1 + \frac{|\text{上年累计折现值}|}{\text{当年折现值}}。$$

表 10-7 各年度的折现系数和折现值 (单位: 万元)

方案	建设期			运营期				
	0	1	合计	2	3	4	5	合计
折现系数	1	0.91		0.83	0.75	0.68	0.62	
甲	年初投资额	350	150	500				
	年末净现金流量				150	200	250	400
	折现值	350	136.5	486.5	124.5	150	170	248
乙	年初投资额	300	200	500				
	年末净现金流量				100	200	300	400
	折现值	300	182	482	83	150	204	248
丙	年初投资额	400	100	500				
	年末净现金流量				200	250	250	300
	折现值	400	91	491	166	187.5	170	186

那么从上表中我们知道

$$(1) \text{ 甲方案的动态投资回收期为: } (5 - 1) + \frac{|-42|}{248} = 4.17 \text{ 年}$$

$$(2) \text{ 乙方案的动态投资回收期为: } (5 - 1) + \frac{|-45|}{248} = 4.18 \text{ 年}$$

$$(3) \text{ 丙方案的动态投资回收期为: } (4 - 1) + \frac{|-137.5|}{170} = 3.81 \text{ 年}$$

建设期：第 0 年折现值为投资支出 (-350)，第 1 年折现值为投资支出 (-136.5)。

运营期：第 2 年折现值 +124.5，第 3 年 +150，第 4 年 +170，第 5 年 +248。

累计折现值计算：第 0 年: -350, 第 1 年: -350 - 136.5 = -486.5, 第 2 年: -486.5 + 124.5 = -362,

第 3 年: $-362 + 150 = -212$, 第 4 年: $-212 + 170 = -42$, 第 5 年: $-42 + 248 = 206$ (首次出现正值), 代入公式就是甲方案的动态投资回收期为: $(5 - 1) + \frac{|-42|}{248} = 4.17$ 年

这里的回收期和静态回收期一样, 也不是绝对时长, 而是一个回收的年份(具体的时间节点)。

这里凯恩解释一下公式, 以甲为例, 就是前 4 年还没回收利润, 到第五年的时候, 还差 42 万, 那么假设全年收入是平均的, 只要 $42/248 = 0.17$ 年就能回本, 所以总得回收公式是 4.17 年。

投资回收率=1/动态投资回收期 × 100%

(1) 甲方案的投资回收率为 $1/4.17 \times 100\% = 23.98\%$ 。

(2) 乙方案的投资回收率为 $1/4.18 \times 100\% = 23.92\%$ 。

(3) 丙方案的投资回收率为 $1/3.81 \times 100\% = 26.25\%$ 。

10.4.5. 典型例题 (重点★★★★★)

(16 年系分) 假设某项目有甲、乙、丙三个解决方案, 投资总额均为 500 万, 建设期均为 2 年, 运营期均为 4 年, 运营期各年末净现金流入量总和为 1000 万, 年利率为 10%, 三种方案的现金流量表如下所示, 试问哪种方案最优? 投资收益率是多少? 净现值率是多少?

表 10-6 三种方案的现金流量 (单位: 万元)

方 案		建设期			运营期				
		0	1	合计	2	3	4	5	合计
甲	年初投资额	350	150	500					
	年末净现金流量				150	200	250	400	1000
乙	年初投资额	300	200	500					
	年末净现金流量				100	200	300	400	1000
丙	年初投资额	400	100	500					
	年末净现金流量				200	250	250	300	1000

首先解释一下, 为什么这里表中的下标都从 0 开始, 建设期的“0 年”常对应初始投资的发生时刻(如工程开工前的资金投入), 属于期初投入。

根据净现值定义，计算出下列矩阵，一般题目都会给你下面这个表，折现不会让你算，因为计算量太大了。我们看第一行折现系数，0 是 1，第 1 年是 $1 / 1.1 = 0.91$ ，以此类推，第 5 年是 $1 / 1.1$ 的 5 次方 = 0.62，也就是说第五年的时候，1 块钱的实际价值相当于现在的 0.62 元（是不是有点理解了，基于的假设是未来的钱是贬值的）。

表 10-7 各年度的折现系数和折现值 (单位: 万元)

方案	建设期			运营期				
	0	1	合计	2	3	4	5	合计
折现系数	1	0.91		0.83	0.75	0.68	0.62	
甲	年初投资额	350	150	500				
	年末净现金流量			150	200	250	400	1000
	折现值	350	136.5	486.5	124.5	150	170	248
乙	年初投资额	300	200	500				
	年末净现金流量			100	200	300	400	1000
	折现值	300	182	482	83	150	204	248
丙	年初投资额	400	100	500				
	年末净现金流量			200	250	250	300	1000
	折现值	400	91	491	166	187.5	170	186

假如从净现值的角度，我们列出下面的表格然后我们求出各个方案的净现值

甲	运营期净现值合计 - 建设期成本净现值合计 = 692.5 - 486.5 = 206 万元
乙	运营期净现值合计 - 建设期成本净现值合计 = 685 - 482 = 203 万元
丙	运营期净现值合计 - 建设期成本净现值合计 = 709.5 - 491 = 218.5 万元

这样看来丙的方案最优（赚的最多）。

再看第二小问投资收益率， $\text{投资收益率} = \text{投资收益} / \text{投资成本} \times 100\%$

(1) 甲方案的投资收益率为 $692.5 / 486.5 \times 100\% = 142.34\%$ 。

(2) 乙方案的投资收益率为 $685 / 482 \times 100\% = 142.12\%$ 。

(3) 丙方案的投资收益率为 $709.5 / 491 \times 100\% = 144.50\%$ 。

所以丙方案的投资收益率是 144.50%。

再看第三问净现值率，按照净现值率就是净现值和投资净现值的比值的定义

$$\text{甲} = 206/486.5 = 42.34\%, \text{乙} = 203/482 = 42.12\%, \text{丙} = 218.5/491 = 44.50\%$$

11.上篇-系统架构设计（次重点★★★☆☆）

质量属性与架构评估系分稍微弱化一点，了解即可。

11.1.软件架构风格（重点★★★★★）

11.1.1.概念辨析（重点★★★★★）

软件架构风格之前案例特别喜欢考，主要是概念的辨析，对比等，难度适中，只要你背了基本都能拿大分。案例简答意思对就行，所以不需要像选择那样记得那么准确，凯恩把红宝书对应的知识化解如下表所示。

一级风格	二级风格	核心概念	典型应用场景
数据流体系结构	批处理	分步执行，前一步完成后整体传递数据，无并行能力	传统数据处理、BAT 脚本
	管道 - 过滤器	前一步输出作为后一步输入，支持并行处理	Unix Shell 程序、编译器阶段处理
调用 / 返回体系结构	主程序 / 子程序	单线程控制，模块化分层调用	早期结构化程序设计
	面向对象	数据与操作封装，通过消息传递交互	Java/C# 应用程序
	客户端 / 服务器	资源分层（表示层 / 功能层 / 数据层），网络交互	Web 系统、数据库应用
以数据为中心体系结构	仓库	中央数据存储，独立构件操作	企业级数据管理系统
	黑板	分级解空间，多领域知识协同	语音识别、模式识别

一级风格	二级风格	核心概念	典型应用场景
虚拟机体系结构	解释器	自定义语言解析执行，支持动态扩展	游戏脚本引擎
	规则系统	基于规则的系统包括规则集、规则解释器、规则/数据选择器及工作内存	专家系统
独立构件体系结构 特殊体系结构	事件驱动	隐式调用，事件广播触发响应	集成开发环境（IDE）、UI 系统
	C2	构件通过连接件分层连接，强制顶部到底部通信	嵌入式系统、分布式应用

11.1.2. 典型例题（重点★★★★★）

(22 年架构真题) 某电子商务公司拟升级其会员与促销管理系统，向用户提供个性化服务，提高用户的黏性。在项目立项之初，公司领导层一致认为本次升级的主要目标是提升会员管理方式的灵活性，由于当前用户规模不大，业务也相对简单，系统性能方面不做过多考虑。新系统除了保持现有的四级固定会员制度外，还需要根据用户的消费金额、偏好、重复性等相关特征动态调整商品的折扣力度，并支持在特定的活动周期内主动筛选与活动主题高度相关的用户集合，提供个性化的打折促销活动。

问：针对该系统的功能，李工建议采用面向对象的架构风格，将折扣力度计算和用户筛选分别封装为独立对象，通过对象调用实现对应的功能；王工则建议采用解释器架构风格，将折扣力度计算和用户筛选条件封装为独立的规则，通过解释规则实现对应的功能。请针对系统的功能，从折扣规则的可修改性、个性化折扣定义灵活性和系统性能三个方面对这两种架构风格进行比较与分析，并指出该系统更适合采用哪种架构风格。

这里题干说明了要你从可修改性、灵活性和系统性能三个方面进行综合考虑。从可修改性来看，解释器风格折扣规则是独立的语法规则，由解释器可对变化的规则进行解析，修改更容易。而面向对象相对固定，有变化需要修改具体的类。从灵活性来看，解释器可以根据用户灵

活解释执行规则，优于面向对象。从系统性能来看，解释器风格一般来说，理论上，软考里，因为多了一层，效率是低于面向对象，这个记住结论即可。综合三个方面来看，解释器的优势更大，所以该系统更适合采用解释器风格。

答：应该选择解释器架构风格。折扣规则的可修改性：解释器风格比面向对象方式实现强。因为解释器风格折扣规则是独立的语法规则，由解释器可对变化的规则进行解析，修改更容易。而面向对象相对固定，有变化需要修改具体的类。个性化折扣定义灵活性：解释器强于面向对象，解释器可以根据用户灵活解释执行规则，做到千人千面。系统性能：面向对象可能略优于解释器。综合来看还是解释器更好。

11.2. 面向架构评估的质量属性（次重点★★★☆☆）

在架构的新教材上先引入了质量属性的概念，然后又讲了面向架构评估的质量属性，但是奇怪的是后者又不是前者的子集，让人十分迷惑。另一方面，这些质量属性又经常混着考，所以凯恩干脆把它们整合到一起。案例主要以选词填空的形式出现，主要目的是看你是否会辨别指定场景到底属于哪个质量属性。

质量属性	定义	衡量指标 / 相关方面
性能	系统的响应能力，即对事件做出响应的时间，或单位时间内处理事件个数	用单位时间内处理事务数量、系统完成事务处理所需时间定量表示，常使用基准测试程序
可靠性	软件系统在意外或错误使用情况下维持功能特性的能力	用平均失效等待时间(MTTF)、平均失效间隔时间(MTBF)衡量；分为容错(错误发生时确保正确行为并内部“修复”)、健壮性(保护程序不受错误使用和输入影响，按预定方式终止执行)
可用性	系统能够正常运行的时间比例	用两次故障之间时间长度、出现故障时恢复正常的速度表示
安全性	系统向合法用户提供服务，阻止非授权用户使用的能力	分为机密性(保证信息不泄露)、完整性(保证信息完整准确)、不可否认性(信息交换双方不能否认行为)、可控性(控制信息传播及内容)

可修改性	快速且高性价比对系统进行变更的能力	可维护性（错误发生后修复软件，局部修改且对其他构件负面影响小）、可扩展性（用新特性扩展软件，松散耦合构件）、结构重组（重新组织构件及关系）、可移植性（适用于多种计算环境）
功能性	系统完成期望工作的能力	依赖系统中许多或大多数构件相互协作
可变性	架构经扩充或变更成为新架构的能力	需符合预先定义规则，与原架构有差异，对软件产品线重要
互操作性	软件与其他系统或自身环境相互作用的能力	为外部可视功能特性和数据结构提供精心设计的软件入口

12.下篇-数据库系统（重点★★★★★）

数据库系统架构案例几乎不考范式，只考性能优化和高可用相关主题。

12.1.数据库性能优化（重点★★★★★）

数据库性能优化的主题软考主要涉及三个一个是索引，一个是数据分区、分片，还有一个就是读写分离，遇到题目你要有话可说。

12.1.1.索引建立注意事项（重点★★★★★）

索引是提高数据库查询速度的利器，而数据库查询往往又是数据库系统中最频繁的操作，因此，索引的建立与选择对数据库性能优化具有重大意义。索引相关知识也是案例常考内容，它的使用准则建议熟记。

索引的建立与选择可遵循以下准则：

(1) 建立索引时，应选用经常作为查询，而不常更新的属性。避免对一个经常被更新的属性建立索引，因为这样会严重影响性能。

(2) 一个关系上的索引过多会影响 UPDATE、INSERT 和 DELETE 的性能，因为关系一

旦进行更新，所有的索引都必须跟着做相应的调整。

(3) 尽量分析出每个重要查询的使用频度，这样，可以找出使用最多的索引，然后可以先对这些索引进行适当的优化。

(4) 对于数据量非常小的关系不必建立索引，因为对于小关系而言，关系扫描往往更快，而且消耗的系统资源更少。

12.1.2. 数据分片（重点★★★★★）

分片（Shard）是把数据库横向扩展（ScaleOut）到多个物理节点上的一种有效的方式，其主要目的是为突破单节点数据库服务器的I/O能力限制，解决数据库扩展性问题。分片（Shard）这个词的意思是“碎片”。如果将一个数据库当作一块大玻璃，将这块玻璃打碎，那么每一小块都称为数据库的碎片。将整个数据库打碎的过程就叫作分片，可以翻译为分片。

12.1.2.1. 分片分类（重点★★★★★）

分片类型是架构设计的起点，决定数据拆分的维度（垂直/水平）。

数据分片	原理
水平分片	水平分片将一个全局关系中的元组分裂成多个子集，每个子集为一个片段。分片条件由关系中的属性值表示。对于水平分片，重构全局关系可通过关系的并操作实现。
垂直分片	垂直分片将一个全局关系按属性分裂成多个子集，满足不相交性。对于垂直分片，重构全局关系可通过连接运算实现。
导出分片	导出分片又称为导出水平分片，即水平分片的条件不是本关系属性的条件，而是其他关系属性的条件。像关系SC，是一个学生选修课表（学号，课程号，成绩）。而是根据学号关联学生表（学号，性别），然后用学生的性别来分片。

混合分片	混合分片是在分片中采用水平分片和垂直分片两种形式的混合。
------	------------------------------

12.1.2.2. 分片算法 (重点★★★★★)

分片算法主要解决了数据被水平分片之后，数据究竟该放在哪个表的问题。案例往往会选择具体的场景来考你，什么样的分片算法适合什么样的数据，看凯恩下面的梳理。

分片算法	原理	适用数据
哈希分片	对指定的键（如 id）进行哈希运算，再根据哈希值对存储节点数量取模，所得余数对应数据应存放的表或节点	数据分布较为均匀，无明显的顺序性或时间相关性；数据之间的关联性较弱，每个数据可独立读写
范围分片	按照特定的范围区间（如时间区间、ID 区间等）来划分数据，将满足某一范围条件的数据存放在一起	具有明显的顺序特征，如按时间先后产生、按编号顺序排列；数据在某个维度上呈现出可划分的区间特性
地理位置分片	依据地理位置信息（如城市、地域等）来分配数据，将同一地理位置相关的数据存储在同一位置或相关联的存储单元中	数据与地理位置紧密相关，不同地理位置的数据使用场景、访问频率等有差异
融合算法	灵活组合多种分片算法，结合不同算法的优势来分配数据	数据特点多样，可能同时具有随机读写、范围查询需求，或兼具多种维度的关联特性

上面的是书本提到的分片算法，但是实际上你落实到具体的一个带分片功能的数据库中间件比如 MyCat，它支持的方式就更多了，这里简单介绍一下最常用的。

分片算法	原理	适用数据
取模算法	对数据的某个键值进行哈希运算，将哈希值对分片数量取模，所得结果作为数据存储的分片索引	适合随机访问的数据，数据分布期望均匀的场景，例如用户 ID、订单 ID 等主键数据
范围算法	按数据的某个字段（如时间、数值等）的范围划分分片	有范围查询需求的数据，例如按时间范围统计的日志数据、按数值区间划分的业务数据等

列表算法	列举数据某个字段的取值，指定每个取值对应的分片	业务场景中取值范围固定且明确的数据，例如地区、部门等有明确分类的数据
一致性哈希算法	将数据键值和节点通过哈希函数映射到固定哈希环上，数据从其哈希值位置顺时针找第一个节点存储；节点增减时，只影响相邻部分数据	分布式系统中节点需要动态调整的数据，例如缓存数据、分布式存储中的对象数据等

12.1.3. 数据分区（重点★★★★★）

数据分区的底层逻辑是将一张表的数据按某种规则（如范围、哈希或复合条件）划分为多个逻辑子集，称为分区。数据分区也在真题中出现过，这里的分区优势、算法特点都要重点记忆。

每个分区存储特定范围或条件的数据。当对分区表执行查询时，数据库查询优化器会根据查询条件和分区规则，确定需要访问哪些分区。数据分区的好处如下表所示。

角度	具体说明
查询性能角度	按特定规则分区，查询时仅访问相关分区数据，减少 I/O 操作，加快查询速度
管理角度	把大表拆分为小且易管理的块，方便数据的维护、备份、恢复等操作
数据隔离角度	可将不同类型或不同时间范围的数据分隔，简化数据管理流程，也利于权限控制等

12.1.3.1. 分区算法（重点★★★★★）

这里的分区算法你会发现和前面分片算法几乎一样。

分区算法	原理	适用数据场景
范围分区	根据数据在某个维度上的连续范围，如日期、数字大小等划分，每个分区存储特定范围的数据	时间序列数据（如按日期存储的日志、交易记录），有明显数据范围区间（如按年龄区间存储的用户信息），方便进行范围查询的数据场景
列表分区	按照离散值列表进行分区，将具有特定离散值的数据划分到同一分区	数据值可枚举且有限的场景，如按地区（固定几个地区选项）、性别（男/女）等维度进行分区的数据

哈希分区	通过哈希函数计算数据的哈希值，依据哈希值将数据分配到不同分区，能让数据较为均匀分布	数据分布需要尽量均匀，对负载均衡要求较高的场景；不太需要范围查询，更注重随机读写性能的场景，如大规模用户数据的存储，可按用户 ID 哈希分区
复合分区	组合使用多种分区类型，先按一种规则进行粗粒度分区，再在每个分区按另一种规则细分	数据量极大且具有多种特征维度的复杂场景，既需要按时间范围（范围分区）又需要按业务类型（列表分区）进行管理的数据

12.1.3.2. 分片与分区的差别（重点★★★★★）

前面谈到了分片和分区，那么它们到底有什么区别。一般来说数据库分片是将数据分散到不同节点，分区是在单个节点内对数据进行分组。

特征	分片	分区
粒度	物理上的划分	物理上的划分
范围	可以跨多个数据库实例或服务器	通常在单个数据库实例内
目的	提高可扩展性和并发性	优化查询性能
实现方式	使用多个物理表	使用 PARTITIONBY 子句
管理难度	相对复杂	相对简单

12.1.4. 分库分表（重点★★★★★）

随着用户量的激增，数据库里的数据越来越多，问题就随之出现了。最典型的三个现象：

第一，资源报警。单机数据库的连接数、IO 和网络吞吐都是有限的，并发量上来之后，数据库就顶不住了。第二，查询变慢。一张表里的数据越来越多，B+树高度变大，访问一次就要更多的 I/O，性能就下降了。第三，热点集中。比如活动期间，某些表或者某些区间会被频繁访问，结果整个数据库都被拖慢。所以我们不得不做分库分表。分库分表的目的其实就是：分库，解决单机承载能力有限的问题，把请求分散到多台服务器上。分表，解决单表数据量过大的问题，把数据拆开，让查询在更小的范围内完成。

12.1.4.1. 分库分表方案（重点★★★★★）

分库和分表都有两种解决方案，分别是水平和垂直，见下表凯恩的概括。

类型	定义	举例场景
水平分表	将同一张表的数据按行划分，分散到多个表中，降低单表数据量	电商订单表 orders 太大，按订单 ID 取模拆成 orders_0、orders_1、orders_2
垂直分表	将一张表的不同列拆分到多个表中，减少字段数量，提高查询效率	商品表 product，基本信息放到 product_base，大字段（描述、图片 URL）放到 product_detail
水平分库	将相同表结构的数据分布到多个数据库实例中，分摊读写压力	社交应用的消息表 message，按用户 ID 拆到 message_db1、message_db2，两个库里都有 message 表
垂直分库	将不同业务的数据分散到不同数据库实例中，按功能模块拆分	在线教育平台：用户数据放 user_db，课程数据放 course_db，支付数据放 payment_db

12.1.4.2. 分库分表 VS 分片分区（重点★★★★★）

分库分表和分片的区别是什么？分库分表是一种物理实现方式，分片是抽象概念。你可以说“我的系统做了分片”，而实现手段就是“分库+分表”。

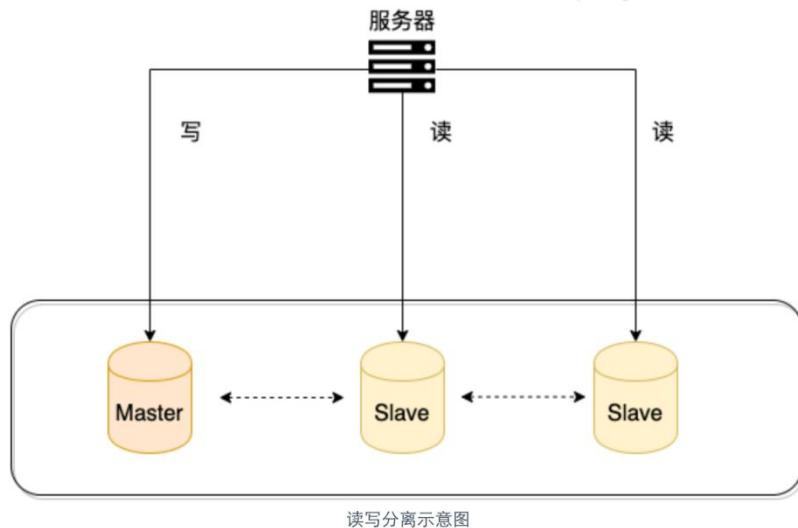
12.2. 数据库高可用模式（重点★★★★★）

前面提到了，数据库的高可用模式是架构考试的常客，读写分离，主从复制都已经考过。

12.2.1. 读写分离（重点★★★★★）

读写分离主要是为了将对数据库的读写操作分散到不同的数据库节点上。这样的话，就能

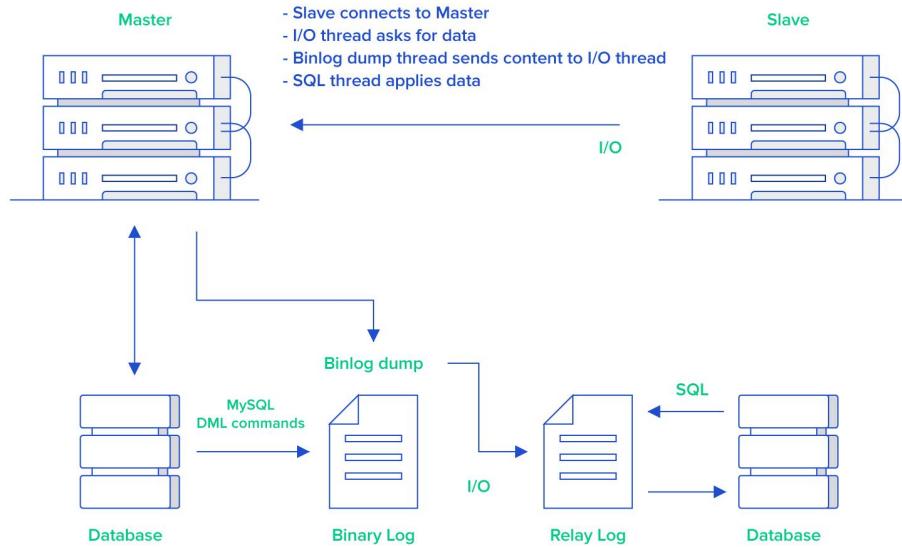
够小幅提升写性能，大幅提升读性能。一般情况下，我们都会选择一主多从，也就是一台主数据库负责写，其他的从数据库负责读。主库和从库之间会进行数据同步，以保证从库中数据的准确性。这样的架构实现起来比较简单，并且也符合系统写少读多的特点。



抛开实践来谈架构没有意义，实际上，Java 生态下可以在应用层使用 Shardsphere，也可以使用数据库中间件 MyCat 在数据库层面去做读写分离的工作。

12.2.2. 主从复制（重点★★★★★）

上一节，凯恩一直在和大家讨论读写分离的场景和实现，但是忽略了一个问题，那就是如何保证主数据库和从数据库之间的数据同步的。其实主从复制利用了日志文件。以 MySQL 为例，MySQL 主要是通过二进制日志（BinaryLog，简称 binlog）实现数据的复制。主数据库在执行写操作时，会将这些操作记录到 binlog 中，然后推送给从数据库，从数据库重放对应的日志即可完成复制。



(1) 主库将数据库中数据的变化写入到二进制日志。

(2) 从库连接主库，并请求复制。

(3) 从库会创建一个 I/O 线程向主库请求更新的二进制日志。

(4) 主库会创建一个二进制日志 dump 线程来发送二进制日志，从库中的 I/O 线程负责接收。

(5) 从库的 I/O 线程将接收的二进制日志写入到中继日志中。

(6) 从库的 SQL 线程从中继日志中读取事件并应用到本地数据库中。

主从复制的优势如下表所示，这块也是需要记忆的。

优势	描述
数据冗余	提供数据副本，防止主库故障导致数据丢失
负载均衡	将读操作分发到从库，减轻主库负载
高可用性	主库故障时，能快速切换到从库，保障系统运行

12.2.3.一致性问题（重点★★★★★）

读写分离对于提升数据库的并发非常有效，但是，同时也会引来一个问题：主库和从库的数据同步往往是存在延迟，比如你写完主库之后，主库的数据同步到从库是需要时间的，这个时间差就导致了主库和从库的数据不一致性问题。这也就是我们经常说的主从同步延迟。一致性问题已经在 2023 年架构设计师考试中考到，需要引起注意，你要掌握解决延迟的三种方法以及优劣势。

同步方式	原理	优点	缺点
半同步复制	主库提交事务后，等至少一个从库接收并写入事务 binlog 到本地中继日志，收到从库 ACK 后主库才正式提交事务并返回客户端。这里涉及到 MySQL 的日志机制感兴趣自己查资料深入了解，软考到这里就已经完事了。	利用数据库原生功能，比较简单	主库写请求时延增长，吞吐量降低
数据库中间件	所有读写走中间件，写请求路由到主库，读请求路由到从库。记录写库 key，主从同步时间窗口内读请求路由到主库，时间过了再路由到从库，如 Canal、Otter。	能保证绝对一致	数据库中间件成本较高
缓存记录	写操作时记录 key 到 cache 并设超时时间（如 500ms）再修改主数据库；读时先查 cache，命中则路由到主库，未命中则路由到从库	相对数据库中间件成本较低	为保证一致性引入 cache 组件，读写数据库时多了缓存操作

拓展阅读

在半同步复制模式下，主库（master）在提交一个事务（即做数据改变操作）后，会等待至少一个从库（slave）接收并写入该事务的二进制日志（binlog）到其本地中继日志（relaylog）中。只有当从库发送确认信息（ACK）给主库后，主库才会正式提交该事务并返回给客户端。

这种机制确保了主从数据的一致性，但以牺牲主库的性能为代价。在 MySQL 中半同步有两种模式有主库先提交，也有主库后提交的，AFTER_SYNC 和 AFTER_COMMIT，看你半同步如何配置，这里大家注意一下就行。

MySQL 的 Replication 默认是一个异步复制的过程，从 MySQL5.5 开始，MySQL 以插件的形式支持半同步复制。半同步复制优点是利用数据库原生功能，比较简单，缺点是主库的写请求时延会增长，吞吐量会降低。

数据库中间件模式。在这种模式下，所有的读写都走数据库中间件，通常情况下，写请求路由到主库，读请求路由到从库。记录所有路由到写库的 key，在主从同步时间窗口内（假设是 500ms），如果有读请求访问中间件，此时有可能从库还是旧数据，就把这个 key 上的读请求路由到主库。在主从同步时间过完后，对应 key 的读请求继续路由到从库。相关的中间件有 Canal，Otter。优点是能保证绝对一致，缺点是数据库中间件的成本较高。

缓存记录（应用层）模式。写流程：如果 key 要发生写操作，记录在 cache 里，并设置“经验主从同步时间”的 cache 超时时间，例如 500ms，然后修改主数据库。相当于你自己做一个中间件。读流程：先到缓存里查看，对应 key 有没有相关数据，如果有相关数据，说明缓存命中，这个 key 刚发生过写操作，此时需要将请求路由到主库读最新的数据。如果缓存没有命中，说明这个 key 上近期没有发生过写操作，此时将请求路由到从库，继续读写分离。这种方式的优点是相对数据库中间件，成本较低，缺点是为了保证“一致性”，引入了一个 cache 组件，并且读写数据库时都多了缓存操作。

12.3. 数据库的选型（重点★★★★★）

文档型数据库、键值对数据库、列存储数据库、图数据库和时序数据库会放在一起考。一个比较常规的出题方向就是给定场景让你挑选合适的数据库，所以你主要掌握不同的数据库产

品到底适合什么业务场景。

分类	场景	结构	产品
键值数据库	内容缓存，处理大量数据的高访问负载，日志系统等	Key 指向 Value 的键值对，通常用 hashtable 实现	Redis
列存储数据库	分布式文件系统，海量数据的高效存储与分析（如大数据场景）	以列簇式存储，将同一列数据存放在一起	Cassandra, HBase
文档型数据库	Web 应用、需要灵活数据结构存储（如电商商品信息管理）	Key-Value 对应的键值对，Value 为结构化数据	MongoDB
图形数据库	社交网络、推荐系统、知识图谱构建（如分析用户关系链）	图结构（节点、边、属性）	Neo4J
时序数据库	物联网设备监控数据存储、工业设备运行状态记录、金融交易数据时序分析等	以时间戳为核心，按时间序列组织数据记录	InfluxDB、TDengine

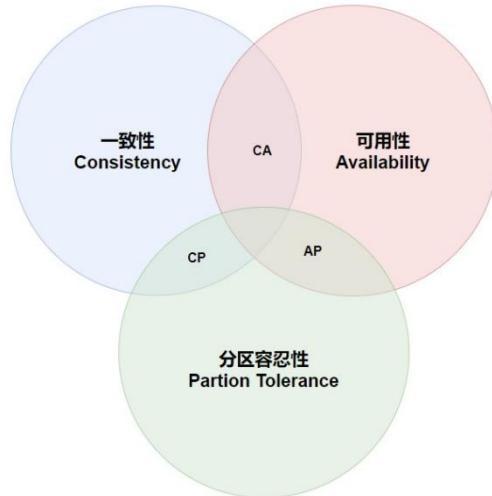
13.下篇-分布式系统（重点★★★★★）

分布式系统目前架构还没考过，这里主要是分为两块，一个是分布式系统的基本理论，另外一块分布式事务的理论解决方案。

13.1.CAP 理论（重点★★★★★）

CAP 理论是指计算机分布式系统的三个核心特性：一致性（Consistency）、可用性（Availability）和分区容错性（Partition Tolerance）。在 CAP 理论中，一致性指的是多个节点上的数据副本必须保持一致；可用性指的是系统必须在任何时候都能够响应客户端请求；而分区容错性指的是系统必须能够容忍分布式系统中的某些节点或网络分区出现故障或延迟。

CAP 理论认为，分布式系统最多只能同时满足其中的两个特性，无法同时满足全部三个特性。

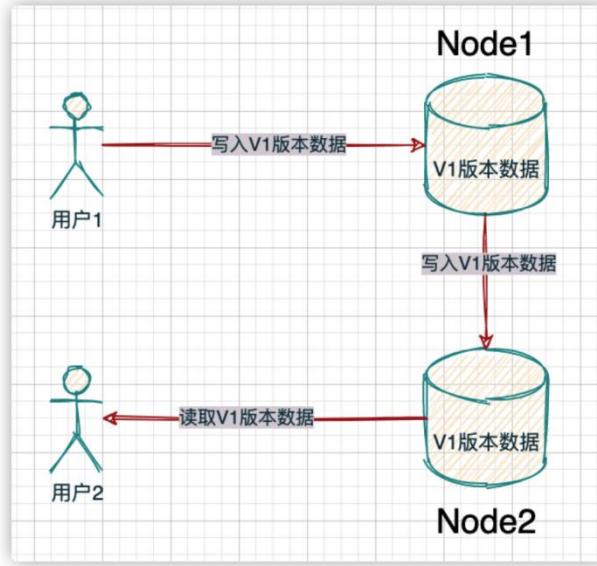


概念	定义
数据一致性	系统中写请求后各相关节点数据均变化，读请求能获取变化后数据
可用性	系统内节点接收读写请求后，都要处理并给出响应结果，即使数据可能有问题
分区容错性	系统中节点通信出现问题产生分区，系统仍需继续运行，各分区可独立对外服务

13.1.1. 数据一致性（重点★★★★★）

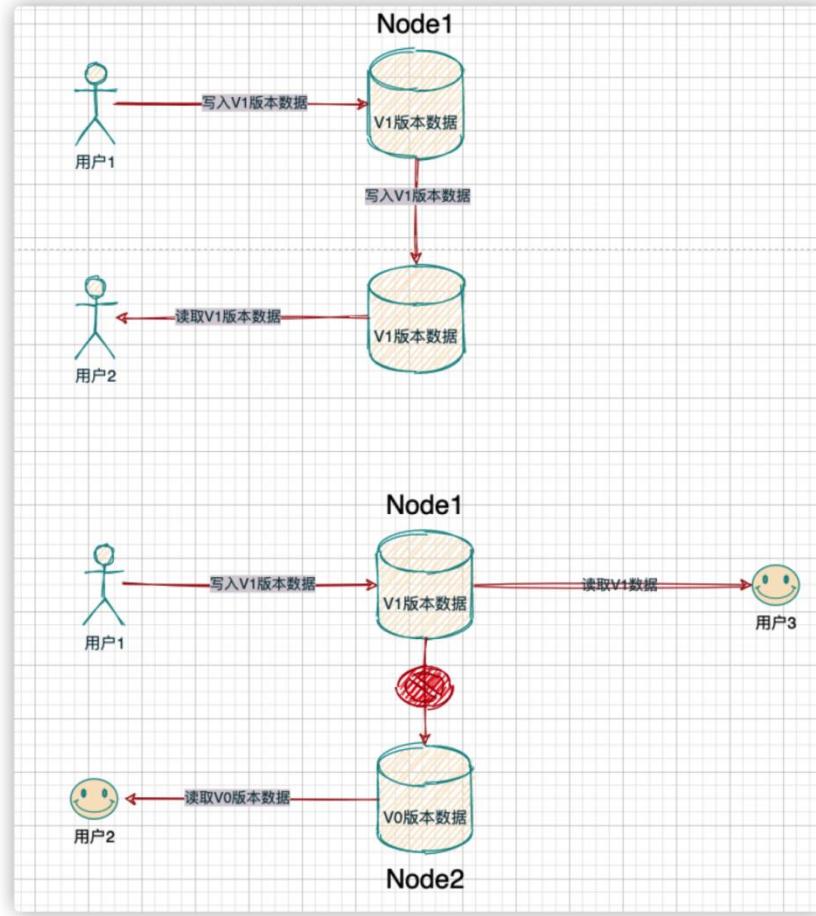
假设，我们的分布式存储系统有两个节点，每个节点都包含了一部分需要被变化的数据。

如果经过一次写请求后，两个节点都发生了数据变化。然后，读请求把这些变化后的数据都读取到了，我们就把这次数据修改称为数据发生了一致性变化。



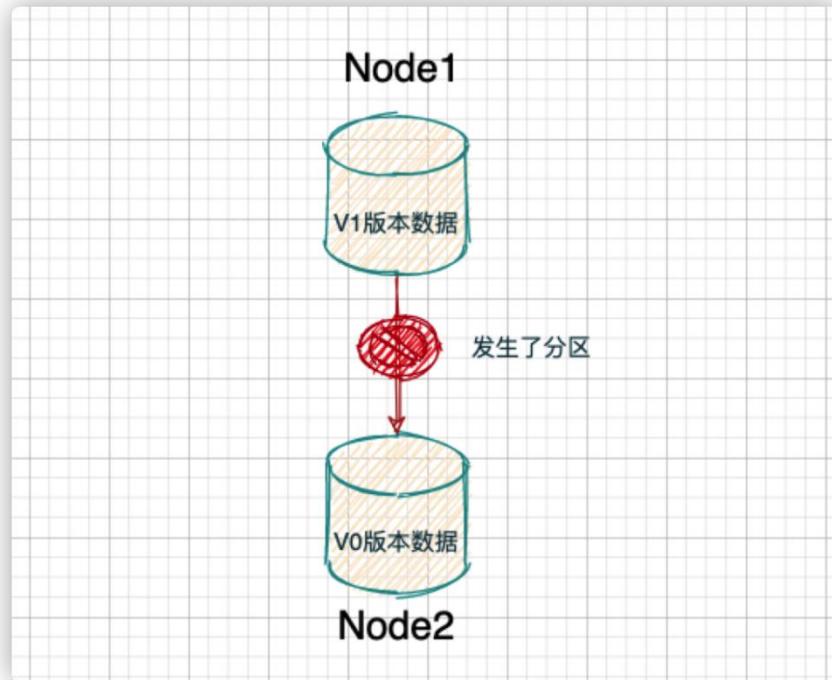
13.1.2. 可用性（重点★★★★★）

如果节点能正常接收请求，但是发现节点内部数据有问题，那么也必须返回结果，哪怕返回的结果是有问题的。比如，系统有两个节点，其中有一个节点数据是三天前的，另一个节点是两分钟前的，如果，一个读请求跑到了包含了三天前数据的那个节点上，抱歉，这个节点不能拒绝，必须返回这个三天前的数据，即使它可能不太合理。

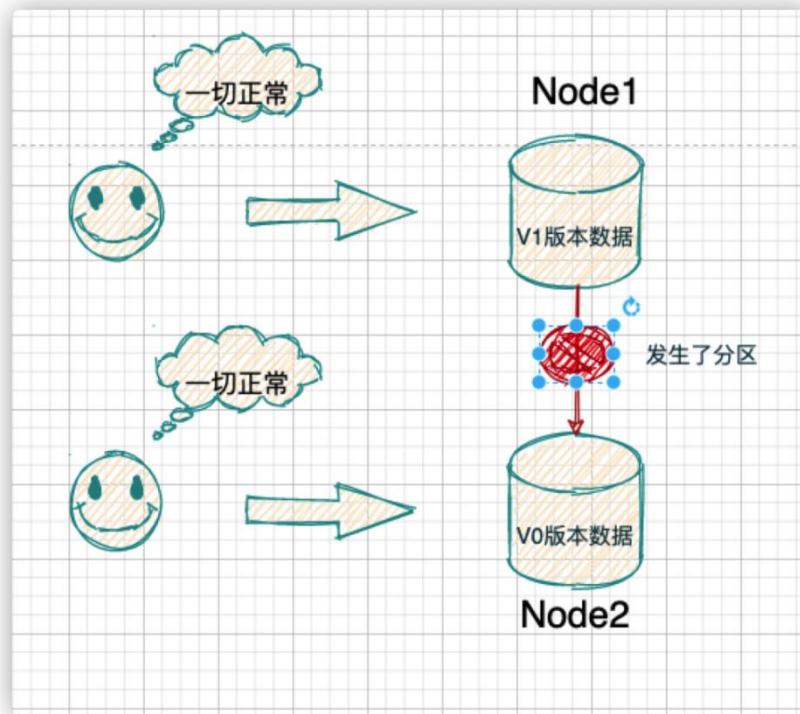


13.1.3.分区容错性（重点★★★★★）

分布式的存储系统会有很多的节点，这些节点都是通过网络进行通信。而网络是不可靠的，当节点和节点之间的通信出现了问题，此时，就称当前的分布式存储系统出现了分区。只要在分布式系统中，节点通信出现了问题，那么就出现了分区。



分区容忍性是指如果出现了分区问题，我们的分布式存储系统还需要继续运行。不能因为出现了分区问题，整个分布式节点全部就停止服务了。



分区容错性和可用性的区别在于，分区容错性本质上是由于通信故障导致分区产生，然后各个分区可以对外独立提供服务。而可用性是指非通信故障下，分布式系统整体对外提供服务

的能力。

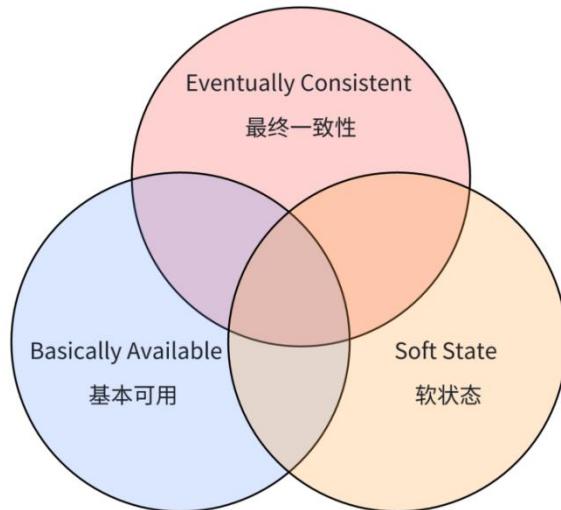
13.2.CAP 理论与系统实现（重点★★★★★）

在分布式系统内，分区容错是必然会发生。我们只能能考虑当发生分区错误时，如何选择一致性和可用性。而根据一致性和可用性的选择不同，开源的分布式系统往往又被分为 CP 系统和 AP 系统。

组合方式	组合问题	例子
一致性和分区容忍性 (CP)	在网络分区期间，某些操作可能不可用，牺牲了可用性。	Zookeeper，如果客户端心跳消失的时候，Zookeeper 会很快剔除该服务，之后就无法提供需求。
可用性和分区容忍性 (AP)	在网络分区期间，不同节点可能有不同的数据副本，导致数据不一致。	Eureka，当 Eureka 客户端心跳消失的时候，Eureka 服务端就会启动自我保护机制，不会剔除该 EurekaClient 客户端的服务，依然可以提供需求。
一致性和可用性 (CA)	无法在分布式环境下实现。	

13.3.BASE 理论（重点★★★★★）

BASE 理论起源于 2008 年，由 eBay 的架构师 Dan Pritchett 在 ACM 上发表。BASE 是对 CAP 中一致性和可用性权衡的结果，其来源于对大规模互联网系统分布式实践的结论，是基于 CAP 定理逐步演化而来的。其基本思路就是：通过业务，牺牲强一致性而获得可用性，并允许数据在一段时间内是不一致的，但是最终达到一致性状态。



在 BASE 理论涉及的三个概念梳理如下表所示。

概念	定义
基本可用	分布式系统遇不可预知故障时，允许损失部分可用性
软状态	允许系统存在中间状态，且该状态不影响整体可用性，不同节点数据副本同步可能存在延时
最终一致性	系统中所有数据副本经一段时间同步后，最终达到一致状态，无需实时强一致性

13.4. 分布式事务理论解决方案

分布式事务是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上。例如在大型电商系统中，下单接口通常会扣减库存、减去优惠、生成订单 id，而订单服务与库存、优惠、订单 id 都是不同的服务，下单接口的成功与否，不仅取决于本地的 db 操作，而且依赖第三方系统的结果，这时候分布式事务就保证这些操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。这一块架构几乎没考过，深入难度比较大。

凯恩这里已经尽力写的想让你能看懂，但对于没有做过开发的同学来说，估计还是很困难。

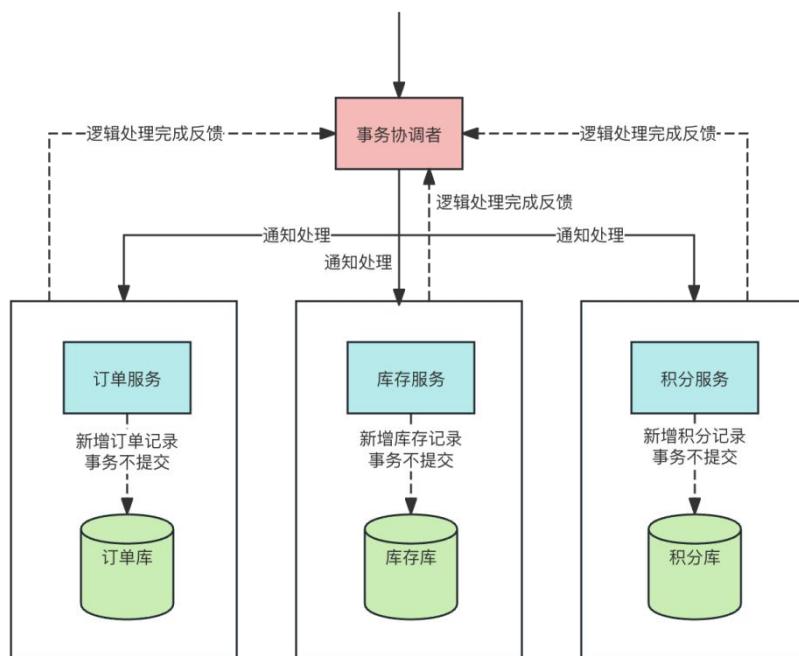
对于这些同学，凯恩建议直接记 2PC TCC 本地消息表的架构图，要能大致描述出方案。

13.4.1. 刚性事务 2PC（重点★★★★★）

2PC 即 Two-Phase Commit，二阶段提交。广泛应用于数据库领域，为了使得基于分布式架构的所有节点可以在进行事务处理时能够保持原子性和一致性。绝大部分关系型数据库，都是基于 2PC 完成分布式的事务处理。顾名思义，2PC 分为两个阶段处理。

阶段一：事务预处理/投票阶段

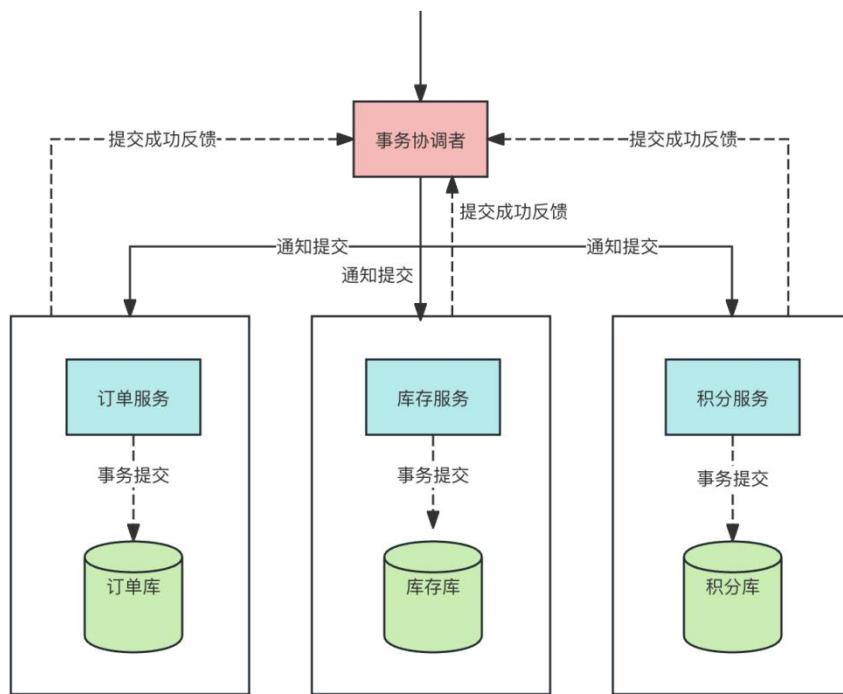
(1) 事务询问。协调者向所有参与者发送事务内容，询问是否可以执行提交操作，并开始等待各参与者进行响应；(2) 执行事务。各参与者节点，执行事务操作，并将 Undo 和 Redo 操作计入本机事务日志；(3) 各参与者向协调者反馈事务问询的响应。成功执行返回 Yes，否则返回 No。



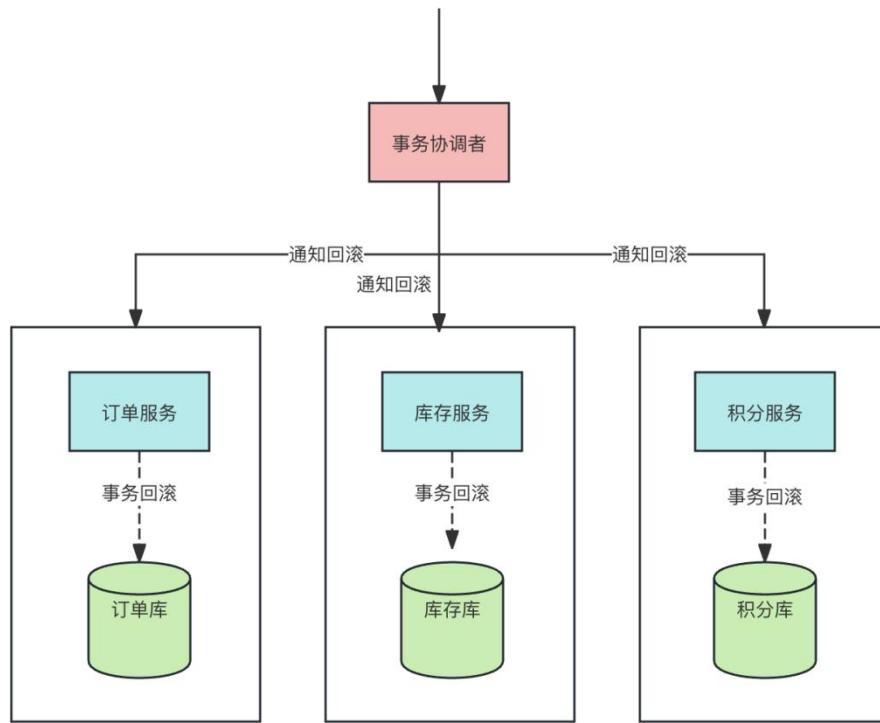
阶段二：提交阶段/执行阶段

协调者在阶段二决定是否最终执行事务提交操作。这一阶段包含两种情形执行事务提交和

中断事务。执行事务提交。所有参与者 Reply Yes，那么执行事务提交。发送提交请求。协调者向所有参与者发送 Commit 请求；事务提交。参与者收到 Commit 请求后，会正式执行事务提交操作，并在完成提交操作之后，释放在整个事务执行期间占用的资源；反馈事务提交结果。参与者在完成事务提交后，向协调者发送 ACK 消息确认；完成事务。协调者在收到所有参与者的 ACK 后，完成事务。



中断事务。事情总会出现意外，当存在某一参与者向协调者发送 No 响应，或者等待超时。协调者只要无法收到所有参与者的 Yes 响应，就会中断事务。发送回滚请求。协调者向所有参与者发送 Rollback 请求；回滚。参与者收到请求后，利用本机 undo 信息，执行 Rollback 操作。并在回滚结束后释放该事务所占用的系统资源；反馈回滚结果。参与者在完成回滚操作后，向协调者发送 ACK 消息；中断事务。协调者收到所有参与者的回滚 ACK 消息后，完成事务中断。



2PC 优点主要体现在实现原理简单，缺点如下

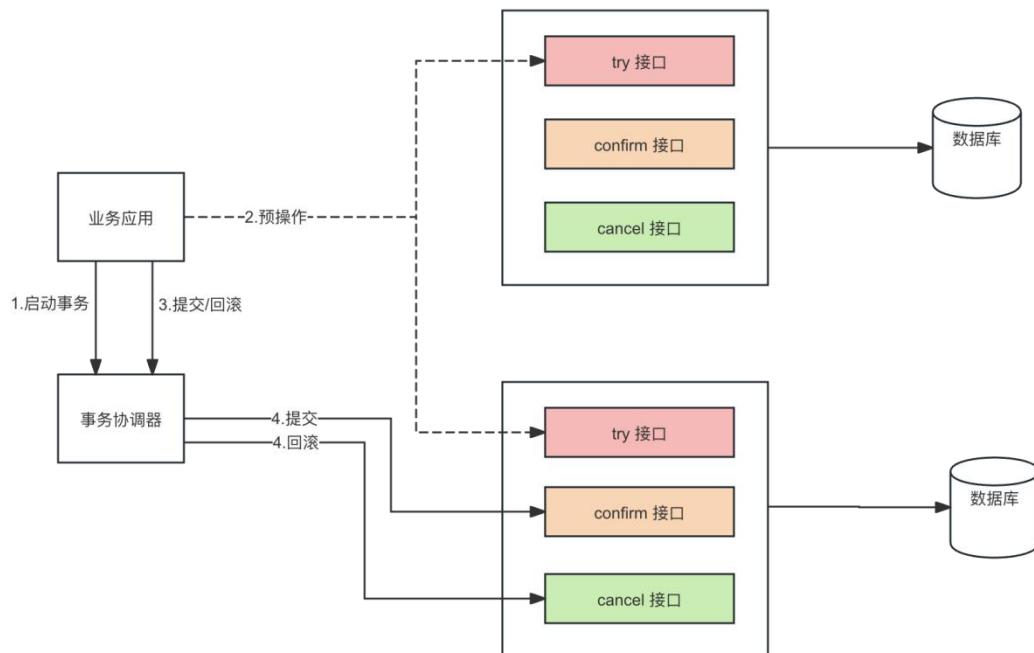
主要问题	解释
性能问题	在 2PC 提交过程中，所有参与事务操作的逻辑都处于阻塞状态，占用系统资源。
单点故障	2PC 协议的协调者是个单点，一旦协调者出现问题，其他参与者将无法释放事务资源，也无法完成事务操作。这是一个严重的可靠性隐患。
数据不一致	当执行事务提交过程中，如果协调者向所有参与者发送 Commit 请求后，发生局部网络异常或者协调者自身崩溃，可能只有部分参与者收到并执行了请求（实际操作只能够超时部分要么做提交要么做回滚）。这会导致整个系统出现数据不一致的情况。
保守机制	2PC 没有完善的容错机制，当参与者出现故障时，协调者无法快速得知，只能严格依赖超时设置来决定是否继续提交或中断事务。这种机制比较保守，无法应对动态的网络环境。

13.4.2. 柔性事务 TCC (重点★★★★★)

关于 TCC (Try-Confirm-Cancel) 的概念，最早是由 Pat Helland 于 2007 年发表的一篇

名为《Life beyond Distributed Transactions:an Apostate's Opinion》的论文提出。

TCC 事务机制相比于 2PC，（1）解决了协调者单点问题，由主业务方发起并完成这个业务活动。业务活动管理器也变成多点，引入集群。（2）同步阻塞：引入超时，超时后进行补偿，并且不会锁定整个资源，将资源转换为业务逻辑形式，粒度变小。（3）数据一致性，有了补偿机制之后，由业务活动管理器控制一致性。



Try 阶段：尝试执行，完成所有业务检查（一致性），预留必需业务资源（准隔离性）。

Confirm 阶段：确认执行真正执行业务，不做任何业务检查，只使用 Try 阶段预留的业务资源，Confirm 操作满足幂等性。要求具备幂等设计，Confirm 失败后需要进行重试。

Cancel 阶段：取消执行，释放 Try 阶段预留的业务资源。Cancel 操作满足幂等性 Cancel 阶段的异常和 Confirm 阶段异常处理方案基本上一致。

在 Try 阶段，是对业务系统进行检查及资源预览，比如订单和存储操作，需要检查库存剩余数量是否够用，并进行预留，预留操作的话就是新建一个可用库存数量字段，Try 阶段操作是对这个可用库存数量进行操作。

基于 TCC 实现分布式事务，会将原来只需要一个接口就可以实现的逻辑拆分为 Try、Confirm、Cancel 三个接口，所以代码实现复杂度相对较高。

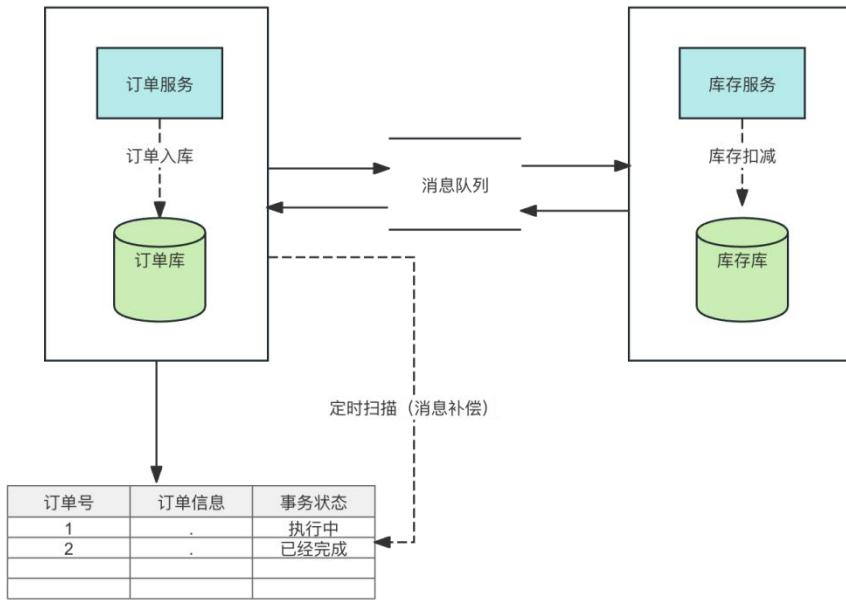
13.4.3. 柔性事务本地消息表（重点★★★★★）

本地消息表这个方案最初是 ebay 架构师 Dan Pritchett 在 2008 年发表给 ACM 的文章。设计核心是将需要分布式处理的任务通过消息的方式来异步确保执行。它通过在本地数据库中记录消息的发送状态，结合异步消息队列，来实现事务的最终一致性。

发送消息方：需要有一个消息表，记录着消息状态相关信息。业务数据和消息表在同一个数据库，要保证它们在同一个本地事务。直接利用本地事务，将业务数据和事务消息直接写入数据库。在本地事务中处理完业务数据和写消息表操作后，通过写消息到 MQ 消息队列。消息投递到 MQ。当消息成功投递到 MQ 后，如果收到消费方返回的 ACK 响应，说明消息被消费方成功处理，此时可以安全地删除本地事务消息表中的记录，以释放资源并避免重复处理。

如果发送失败，也就是没有收到 ACK 响应，确实需要进行重试。重试的目的是确保消息最终能够被消费方成功处理，从而保证分布式事务的一致性。可以通过设置重试次数和重试间隔等策略来控制重试过程，同时要注意处理可能出现的重复消息问题，比如消费方需要具备幂等性，以确保即使重复处理消息也不会产生错误的结果。

消息消费方：处理消息队列中的消息，完成自己的业务逻辑。如果本地事务处理成功，则表明已经处理成功了。如果本地事务处理失败，那么就会重试执行。如果是业务层面的失败，给消息生产方发送一个业务补偿消息，通知进行回滚等操作。



本地消息表实现了分布式事务的最终一致性，优缺点比较明显。优点：实现逻辑简单，开发成本比较低。缺点：与业务场景绑定，高耦合，占用业务系统资源，量大可能会影响数据库性能。由于消息是异步发送和处理的，可能会引入一定的延迟。

14.后记

这本《芝士架构冲刺红宝书》作为备考冲刺工具，它在考点梳理、真题解析和表格速记方面确实下了功夫。但是因为凯恩水平有限，尽管已全力以赴，但这份红宝书依然存在诸多不足之处。虽然已经有近 9 万字，但是它无法覆盖所有的考点。比如红宝书里关于真题的解析多集中在需求分析、数据库等常考知识点，对嵌入式系统、物联网等方向的案例涉及较少。也许，即使你把这里的内容全部背出，上考场，你仍然会看到很多新技术新考法，让你做得很难受。

凯恩不可能把所有的技术相关的内容都涵盖，有些东西的确要靠自己的积累。希望你有背出这本红宝书的实力+也有会什么考什么的运气。如果大家在使用过程中有任何疑问或建议，欢迎随时联系微信 Deckardcain2 与我交流。