One page note for system architecture designer

绪论

1. **架构**：架构是系统中组件的基本组织方式，涵盖了组件之间的关系、与环境的关系，以及指导系统设计和发展的原则。
2. **系统**：系统是由组件组成的集合，用于完成特定的功能。它可以是一个单独的应用程序、一个完整的系统、

子系统、系统之系统、产品线、整个企业或任何其他感兴趣的集合。

3. **环境**：环境或上下文决定了系统的开发、运行、政策以及对其产生影响的外部条件和设置。
4. **任务**：任务是由一个或多个利益相关者通过系统来实现特定目标的操作或用途。

系统架构（System Architecture）

- 系统架构是系统整体高层次的结构表示。
- 它是系统的骨架和根基，支撑和链接各个部分，包括组件、连接件、约束规范以及指导这些内容设计与演化的原理。
- 系统架构是刻画系统整体抽象结构的一种手段。
- 架构设计的目的是对系统进行一系列相关的抽象，以指导系统的设计与实现。
- 架构设计在系统开发过程中起着关键作用，决定了系统的健壮性和生命周期的长短。

架构设计的作用：

1. 解决复杂的需求分析问题。
2. 解决**非功能属性**在系统设计中的重要问题。
3. 解决生命周期长、扩展性要求高的系统整体结构问题。
4. 解决系统基于组件的集成问题。
5. 解决业务流程再造的问题。

软件架构发展阶段

基础研究阶段（**模块化开发方法**：高内聚、低耦合、模块大小适度、接口简单）----> 概念体系和核心技术形成（**组件化技术**： 可组装性和可插拔性、组件化比模块化更独立、比应用集成结合的更加紧密）----> 理论体系完善与发展（软件架构描述与表示、软件架构分析、设计与测试；软件架构发现、演化与重用；软件架构开发方法；软件架构风格；动态软件架构）----> 普及应用阶段（**软件架构是软件生命周期的重要产物**，影响各个阶段）

软件架构的常用分类和建模方法

1. 分层架构：TCP-IP协议栈、表现层-业务层-持久层-数据库
2. 事件驱动架构：事件进行通信，分布式异步架构模式，适用于松散耦合系统。事件队列->分发器->事件通道->事件处理器
3. 微核架构：插件架构，Core只包含系统运行的最小功能，插件则互相独立来实现具体业务逻辑
4. 微服务架构：每一个服务就是一个独立的部署单元。RESTful API模式、RESTful应用模式、消息集中模式（Message Broker）
5. 云架构：基于云平台的架构，将应用程序和数据部署在云平台上，实现弹性扩展、高可用、高可靠、高安全。（处理单元、虚拟中间件）

软件架构的模型：这4种模型并非完全独立，要结合不同模型才能准确、全面地反映软件架构。

1. **结构模型**：是*最直观*、*普遍的*建模方法，通过架构的*构件*、*连接件和其他概念*刻画结构，力求反映系统的配置、约束、隐含假设条件、风格和性质等重要语义内容，研究其的核心是*架构描述语言*。
2. **框架模型**：和结构模型类似，但不太侧重描述结构细节，更侧重整体结构，主要以一些特殊问题为目标建立只针对和适应问题的结构。
3. **动态模型**：是对结构或框架模型的补充，主要研究*系统"大颗粒"行为的性质*，比如描述系统的重新配置或演

化，这里的动态可指系统总体结构的配置、建立或拆除通信或计算的过程，这类系统模型常是激励型的。

4. **过程模型**：研究*构造系统的步骤和过程*，其结构是遵循某些过程脚本的结果。

软件架构应用场景及未来发展

架构风格	适用场景
管道/过滤器风格	用于将系统分成若干独立的步骤
主程序/子系统和面向对象的架构风格	用于对组件内部进行设计
虚拟机风格	用于构造解释器或专家系统
C/S 和 B/S 风格	适合于数据和处理分布在一定范围，通过网络连接构成系统
平台/插件风格	用于具有插件扩展功能的应用程序
MVC 风格	用于用户交互程序的设计
SOA 风格	用在企业集成等方面
C2 风格	用于 GUI 软件开发，用以构建灵活和可扩展的应用系统等

架构发展主线可归纳为模块化编程、面向对象编程、构件技术、面向服务开发技术和云技术。

计算机基础知识

计算机系统概述——关键部件介绍

处理器（CPU）

- **核心地位**：是计算机系统运算与控制的核心部件。
- **指令集分类**
    - **复杂指令集（CISC）**：以Intel、AMD的x86CPU为代表。
    - **精简指令集（RISC）**：以ARM和Power为代表。
- **其他相关处理器**
    - **图形处理器（GPU）**：属于特殊类型处理器，拥有数百或数千个内核，经优化可并行运行大量计算，在深度学习和机器学习领域应用广泛。
    - **信号处理器（DSP）**：专门用于实时的数字信号处理。
    - **现场可编程逻辑门阵（FPGA）**：也属于处理器相关范畴。

存储器

- **基本定义**：利用半导体、磁、光等介质制成的电子设备，用于存储数据。
- **按硬件结构分类**：可分为SRAM、DRAM、NVRAM、Flash、EPROM、Disk等。
- **存储体系结构（按与处理器物理距离分）**
    - **片上缓存**：在处理器核心中直接集成的缓存，采用SRAM结构，容量小，速度快。

    - **片外缓存**：在处理器核心外的缓存，需经交换互联开关访问，由SRAM构成，容量较片上缓存略大，可称为L2Cache、L3Cache或平台Cache（Platform Cache）。
    - **主存（内存）**：采用DRAM结构，以独立的部件/芯片存在，通过总线与处理器连接。
    - **外存**：如硬盘、光盘等设备，这类设备访问速度慢，但容量大，且掉电后能保持数据。

层次化存储结构

存储器定义与分类：存储器是计算机系统中的记忆设备，用来存放程序和数据。存储器包括主存、辅存、通用寄存器、Cache四类。

各存储层级特点：

- **CPU内部通用寄存器**：容量小、速度快但成本高。
- **Cache**：按照内容存取，主存到Cache的映射由硬件完成。
- **主存储器**：分为RAM和ROM两类。
- **联机磁盘存储器**：属于辅存层级。
- **脱机光盘、磁盘存储器**：属于辅存层级。

层级规律与理论依据：

- 由上到下速度越来越慢，成本越来越低，容量越来越大。
- 局部性原理是层次化存储结构的支撑和理论依据。
- 主存和辅存构成虚拟存储器。
- 目前计算机三级存储体系为Cache、主存和辅存。

Cache

为了解决高速运行的CPU与主存储器之间速度不匹配的问题，Cache中存放的是主存的部分拷贝（副本），是按照程序的局部性原理选取出来的最常使用或不久将来仍将使用的内容。

- **Cache的存储层级地位**：在计算机的存储系统体系中，Cache是访问速度最快的层次（如果题目中有寄存器选寄存器，如果无选Cache）。
- **Cache改善性能的依据**：程序的局部性原理，包含时间局部性和空间局部性。
    - **时间局部性**：程序中的某条指令一旦执行，不久以后该指令可能再次执行，例如程序中存在大量的循环操作。
    - **空间局部性**：一旦程序访问了某个存储单元，不久以后，其附近的存储单元也被访问，即程序在一段时间内所访问的地址可能集中在一定的范围内，例如顺序执行或数组等。
- **命中与不命中**：CPU访问内存时，若所需内容在Cache中称为“命中”，直接从Cache调用；否则称为“不命中”，需从主存调用。CPU可直接从Cache读写内容，因其存取速率快，能提高CPU利用率和系统性能。
- **平均访问时间公式**：若h为Cache访问命中率，t_1为Cache存取时间，t_2为主存访问时间，则平均访问时间$t_a = h \times t_1 + (1-h) \times t_2$。

磁盘

- **磁盘格式化存储容量公式**：格式化存储容量$= n \times t \times s \times b$，其中$n$为保存数据的总记录面数，$t$为每面磁道数，$s$为每道的扇区数，$b$为每个扇区存储的字节数。
- **磁盘非格式化存储容量计算方式**：记录面数×（内直径周长×位密度）×内外半径的磁道数。

- **硬盘存取时间构成**：寻道时间＋等待时间＋读/写时间，其中读/写时间可忽略不计，通常以平均寻道时间＋平均等待时间来衡量。等待时间也叫旋转延迟

磁盘寻到算法：

- **先来先服务（FCFS）**：按照请求的先后顺序处理磁盘访问请求，简单但可能导致寻道时间较长。
- **最短寻道时间优先（SSTF）**：优先处理与当前磁头位置寻道时间最短的请求，能减少寻道时间，但可能导致某些请求长期等待（饥饿）。
- **扫描算法（SCAN）**：磁头沿一个方向移动，处理该方向上的所有请求，直到到达磁道的一端，然后反向移动处理剩余请求，类似电梯运行，可避免饥饿。
- **循环扫描（CSCAN）算法**：磁头沿一个方向移动，处理该方向上的所有请求，到达磁道一端后直接回到另一端的起始位置，再次沿原方向处理请求，使等待时间更均匀。

总线

- **定义**：计算机部件间遵循特定协议，以特定格式和控制逻辑实现数据交换与传输的形式。
- **分类（按在计算机中的位置）**
  - **内总线**：用于各类芯片内部互连，也可称为片上总线（On - Chip Bus）或片内总线。
  - **系统总线**：计算机中CPU、主存、I/O接口的总线。
  - **外部总线**：计算机板与外部设备之间，或计算机系统之间互联的总线，又称通信总线。
- **性能指标**：常见的有总线带宽、总线服务质量QoS、总线时延和总线抖动等。
- **类型**
  - **并行总线**：主要包括PCI、PCIe、ATA（IDE）等。
  - **串行总线**：主要包括USB、SATA、CAN、RS - 232、RS - 485、RapidIO和以太网等。

接口

- **定义**：同一计算机不同功能层之间的通信规则。
- **常见类型**：显示类接口（HDMI、DVI等）、音频输入输出类接口（TRS、RCA、XLR等）、网络类接口（RJ45、FC等）、PS/2接口、USB接口、SATA接口、LPT（打印接口）、RS - 232接口等。

外部设备

- **常规设备**：键盘、鼠标、显示器、扫描仪、摄像头、麦克风、打印机、光驱、各型网卡、各型存储卡/盘等。
- **移动与穿戴设备**：加速计、GPS、陀螺仪、感光设备、指纹识别设备等。
- **工业等特殊领域设备**：测温仪、测速仪、轨迹球、各型操作面板、红外/NIFC等感应设备、各种场强测量设备、功率驱动装置、各型机械臂、各型液压装置、压装装置等。

计算机软件

软件系统是在计算机硬件系统上运行的**程序、相关文档资料和数据的集合**，作用是扩充计算机系统功能、提高系统效率。

计算机软件通常分为**系统软件**和**应用软件**两大类。

操作系统

操作系统组成：由操作系统内核（Kernel）和诸多附加配套软件组成，附加软件包括图形用户界面程序、常用应用程序以及支持应用软件开发和运行的各种软件构件（如应用框架、编译器、程序库等）。其中，操作系统内核是能提供进程管理（任务管理）、存储管理、文件管理和设备管理等功能的软件模块，是操作系统最基本部分，驻留内存，以CPU最高优先级运行，可执行特权指令，能直接访问外设和全部主存空间，负责系统资源管理与分配。

操作系统的作用：管理计算机中运行的程序并分配各种软硬件资源；为用户提供友善的人机界面；为应用程序的开发和运行提供高效率平台。此外，还具有辅导用户操作（帮助功能）、处理软硬件错误、监控系统性能、保护系统安全等作用。

操作系统的特征:

- **并发性**：多道程序环境下，一段时间内宏观上多个程序同时运行，但单CPU环境下，每一时刻只有一个程序在执行。
- **共享性**：操作系统中的资源被多个并发执行的进程共同使用，而非被一个进程独占。
- **虚拟性**：把一个物理实体变成逻辑上的多个对应物，或把物理上的多个实体变成逻辑上的一个对应物的技术。
- **不确定性**：多道程序环境中，进程走走停停，执行并非一贯到底。

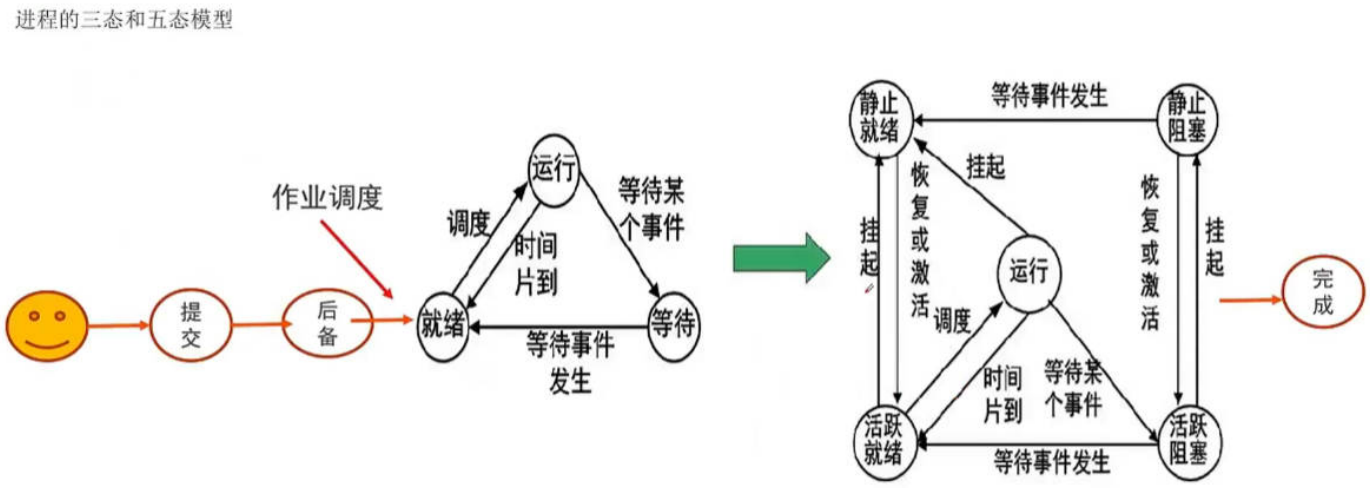
操作系统的分类：

- **批处理操作系统**：分为单道批处理和多道批处理。单道批处理是用户可向系统提交多个作业，但一次只有一个作业装入内存执行，一个结束后另一个开始；多道批处理操作系统允许一次多个作业装入内存执行，任意时刻作业都处于开始点和终止点之间，具有多道、宏观上并行运行和微观上串行运行的特点。
- **分时操作系统**：将CPU的工作时间划分为很多很短的时间片，轮流为各个终端的用户服务，具有**多路性、独立性、交互性和及时性**的特点。
- **实时操作系统**：包含实时控制系统和实时信息系统，能对输入的信息快速处理，并在**被控对象允许的时间范围内做出快速反应**，对可靠性有保障要求。
- **网络操作系统**：是方便且有效地共享网络资源，为网络用户提供各种服务的软件和有关协议的集合。其特征包括**硬件独立性**（可运行在不同网络硬件上，能通过网桥或路由器与其他网络连接）和**多用户支持**（能同时支持多个用户对网络的访问，对信息资源提供完全的安全和保护功能）等。
- **分布式操作系统**：由多个分散的计算机经连接而成的计算机系统，系统中的计算机无主、次之分，任意两台计算机可通过通信交换信息。它是网络操作系统的更高级形式，保持网络系统全部功能的同时，还具有透明性、可靠性和高性能等特性。
- **微型计算机操作系统**：简称微机操作系统，常用的有Windows、Mac OS、Linux。
- **嵌入式操作系统**：运行在嵌入式智能设备环境中。对整个智能硬件以及它所操作、控制的各种部件装置等资源进行统一协调、处理、指挥和控制。
    - **微型化**：从性能和成本角度考虑，希望占用的资源和系统代码量少，如内存少、字长短、运行速度有限、能源少（用微小型电池）。
    - **可定制**：从减少成本和缩短研发周期考虑，能运行在不同的微处理器平台上，能**针对硬件变化进行结构与功能上的配置**，以满足不同应用需要。
    - **实时性**：主要应用于过程控制、数据采集、传输通信、多媒体信息及关键要害领域需要**迅速响应**的场合，对实时性要求较高。
    - **可靠性**：系统构件、模块和体系结构必须达到应有的可靠性，对关键要害应用还要提供容错和防故障

措施。

- **易移植性**：为提高易移植性，通常采用硬件抽象层（Hardware Abstraction Level，HAL）和板级支撑包（Board Support Package，BSP）的底层设计技术。

进程状态模型



三态模型

- **运行**：进程已获取必需资源，正在处理机上执行。
- **就绪**：进程已获得除CPU外的所有必需资源，一旦获取CPU就会立即运行。
- **等待（阻塞）**：进程因等待某事件发生（如I/O完成）而暂时无法执行。

五态模型（引入外部存储）

- **运行**：同三态模型的运行状态。
- **活跃就绪**：进程在内存中，具备运行条件，等待调度。
- **活跃阻塞**：进程在内存中，因等待事件而阻塞。
- **静止就绪**：进程被挂起至外存，具备运行条件，等待激活后进入活跃就绪。
- **静止阻塞**：进程被挂起至外存，因等待事件而阻塞，等待激活后进入活跃阻塞。

调度类型

- **高级调度（作业调度）**：批处理系统中，决定作业何时从外存进入内存。分时和实时操作系统通常不需要。
- **中级调度（内存调度）**：实现内存与外存的进程交换，用于五态模型的挂起/激活。
- **低级调度（进程调度）**：决定就绪队列中的进程何时获得CPU，对应三态模型的状态切换。

信号量

进程间制约关系

- 间接相互制约关系：源于资源共享（临界资源）。
- 直接相互制约关系：源于进程合作（进程之间的顺序或逻辑关系）。
  临界资源：一段时间内只允许一个进程访问的资源，如打印机，进程间需互斥访问。

信号量操作（PV操作）

- **P操作（申请资源，减量操作）**
  - 将信号量S的值减1，即$S = S - 1$。
  - 若$S \geq 0$，进程继续执行；否则进程置为等待状态。
- **V操作（释放资源，增量操作）**
  - 将信号量S的值加1，即$S = S + 1$。
  - 若$S > 0$，进程继续执行；否则唤醒等待队列中的等待进程。

信号量的应用

- **进程互斥**
  - 为临界资源设置互斥信号量（`mutex`），初始值为1（有n个临界资源则设为n）。
  - 进程访问临界资源前执行P操作，访问后执行V操作，保证不同时进入临界区。
- **进程同步**
  - 运行条件不满足时，进程执行P操作暂停；条件满足时，执行V操作继续。
  - 例如生产者-消费者问题：通过信号量控制缓冲区的空满状态，保证生产和消费的协调。
  - **不能向满队列添加，需要一个初值为队列容量的信号量；不能向空队列拉取，需要一个初值为0的信号量；不能同时操作队列，需要一个互斥信号量**
- **进程前驱关系**
  - 前驱图是有向无循环图，节点代表程序段操作，有向边表示前趋关系，用于描述进程执行顺序。
  - 做题法门：有几个有向边就设置几个信号量，初始值为0；按顺序将信号量放在有向边上；进程执行前进行P操作，执行后进行V操作。

死锁相关

- **保持和等待**：一进程在请求新的资源的同时，保持对已分配资源的占用。
- **不可剥夺**：进程已经获得的资源，在未使用前不能被剥夺，只能使用完成自己释放。
- **环路等待**：发生死锁时，必然存在一个进程、资源的环形链。
- **互斥**：临界资源的访问，即一段时间内某个资源只有一个进程占用，如果此时还有其他进程请求该资源，则请求者只能等待，直到占有该资源的进程完全释放。

死锁资源计算

- 发生死锁的最大资源数为$n \times (R - 1)$，表示每个进程都差一个资源就能满足需求，此时系统资源耗尽且进程相互等待，形成死锁。
- 不发生死锁的最小资源数在此基础上加1，确保至少有一个进程能获得足够资源完成执行，释放资源后其他进程也能顺利执行。

避免死锁的方式

- **预防死锁**：通过打破死锁产生的四个必要条件来实现，例如采用资源的静态分配策略等。
- **避免死锁**
  - 有序资源分配法：将资源按顺序编号，进程按编号递增的顺序请求资源，可避免环路等待。
  - 银行家算法：通过预先模拟资源分配后的状态，判断是否存在安全序列，从而避免死锁。

- **检测死锁**：允许系统在运行过程中发生死锁，通过设置检测机构及时检测死锁的发生，确定与死锁有关的进程和资源。
- **解除死锁**：是死锁检测的配套措施，当检测到死锁时，通过撤销或挂起进程回收资源，再将资源分配给其他进程，使系统摆脱死锁状态。常用方法有撤销进程、挂起进程等。

页式存储

定义：将程序与内存划分成同样大小的块，以页为单位将程序调入内存。

- **地址转换**
  - 逻辑地址：由页号和页内地址组成，即逻辑地址=页号+页内地址。
  - 物理地址：由页帧号（物理块号）和页内地址组成，即物理地址=页帧号+页内地址。
  - 转换依据：通过页表实现逻辑页号到物理页帧号的映射。
- **优缺点**
  - 优点：内存利用率高，碎片小，分配及管理简单。
  - 缺点：增加了系统开销（如页表管理、地址转换），可能产生抖动现象（频繁的页面换入换出）。

页面淘汰算法知识拓展（先淘汰未被访问，再淘汰未被修改的）

- 淘汰原则的核心是利用程序局部性原理，优先淘汰“不常用且修改代价小”的页面，以减少页面换入换出的开销，避免抖动现象。
- 未被访问的页面大概率近期也不会被访问，未修改的页面与辅存内容一致，换出时无需写回辅存，代价更小。

段式存储

分段式存储管理系统中，为每个段分配一个连续的分区，进程中的各个段可离散地分配到主存的不同分段中。

- **段表作用**：系统为每个进程建立段映射表（段表），每个段在表中占一个表项，记录该段在主存中的起始地址（基址）和段的长度。进程执行时，通过查段表找到每个段对应的主存区。
- **逻辑地址结构**：由段号S和段内地址d（偏移量）组成。
- **地址转换步骤**
  1. 检查段号S与段表长度L，判断是否地址越界。
  2. 通过段表起始地址和段号S，查找段表中对应的表项，获取段长L和基址。
  3. 检查段内地址d与段长L，判断是否段内越界。
  4. 若均未越界，物理地址 = 基址 + 段内地址d。

段页式存储

是段式和页式的结合体，先将用户程序分成若干段，再把每个段分成若干页，并为每个段赋予一个段名。每个页的大小相同，但每个段的大小不同。

逻辑地址由段号、段内页号、页内地址组成。

地址转换步骤

1. 检查段号与段表长度，判断是否地址越界。
2. 在段表寄存器中找到段表起始地址，查找段表中对应段的表项，获取页表起始地址。

3. 检查页号与页表长度，判断是否页内越界。
4. 查找页表中对应页的表项，获取物理块地址。
5. 物理地址 = 物理块地址 + 页内地址。

- 优点：空间浪费小，存储共享容易，存储保护容易，能动态连接。
- 缺点：管理软件复杂，开销增加，需要的硬件和内存占用增加，执行速度下降。

流水线计算

公式

- 理论公式：$(t_1 + t_2 + \cdots + t_k) + (n-1) \times \Delta t$（$k$为流水线分段数，$n$为指令条数，$\Delta t$为流水线周期）。
- 实践公式：$(k + n - 1) \times \Delta t$。

流水线周期：执行时间最长的一段。

吞吐量（TP）：$TP = \frac{\text{指令条数}}{\text{流水线执行时间}}$。

最大吞吐量（$TP_{\max}$）：$TP_{\max} = \lim_{n\to\infty} \frac{n}{(k+n-1)\Delta t} = \frac{1}{\Delta t}$。

加速比：$S = \frac{\text{不使用流水线执行时间}}{\text{使用流水线执行时间}}$，用于衡量并行系统或程序并行化的性能效果。

超标量流水线：一般为m个度的流水线，相当于几条流水线同时运作。

流水线的效率：流水线的设备利用率，在时空图上，为N个任务占用的时空区与K个流水段总时空区之比。

例题分析（以取指2ns、分析2ns、执行1ns，100条指令为例）

- **流水线周期**：2ns（取指或分析的最长时间）。
- **流水线执行时间**
  - 理论公式：$(2+2+1) + (100-1) \times 2 = 203$ ns。
  - 实践公式：$(3 + 100 - 1) \times 2 = 204$ ns。
- **不采用流水线顺序执行时间**：$(2+2+1) \times 100 = 500$ ns。
- **加速比**：$500/203 \approx 2.46$。
- **吞吐量**：$100/203 \approx 0.49$，最大吞吐量：$1/2 = 0.5$。
- **度为5的超标量流水线执行时间**：每条流水线执行$100/5 = 20$条指令，执行时间为 $(2+2+1) + (20-1) \times 2 = 43$ ns，若不能整除需向上取整。

缓冲区计算

某计算机系统输入/输出采用双缓冲工作方式，假设磁盘块与缓冲区大小相同，每个盘块读入缓冲区的时间T为$10\ \mu$s，缓冲区送用户区的时间M为$6\ \mu$s，系统对每个磁盘块数据处理时间C为$2\ \mu$s。若用户需要将大小为10个磁盘块的Doc1文件逐块从磁盘读入缓冲区，并送用户区进行处理，那么采用双缓冲需要花费的时间为（）μs，比使用单缓冲节约了（）μs时间。

A. 100 B. 108 C. 162 D. 180
A. 0 B. 8 C. 54 D. 62

- **双缓冲执行时间**：

双缓冲下，读入第1个磁盘块后，读下一个磁盘块与前一个磁盘块的传送、处理可并行。
时间计算为：$10 \times 10 + 6 + 2 = 108\ \mu s$，所以第一空选 **B**。

- **单缓冲执行时间**：
  单缓冲下，每个磁盘块的读入、传送、处理需串行部分操作，时间为：$(10 + 6) \times 10 + 2 = 162\ \mu s$。
  节约时间：$162 - 108 = 54\ \mu s$，所以第二空选 **C**。
- **双缓冲**：通过两个缓冲区实现I/O操作与数据处理的并行，提高系统效率，适用于I/O设备与CPU处理速度差异较大的场景。
- **单缓冲**：仅一个缓冲区，I/O操作与数据处理串行度高，效率低于双缓冲。

指令集

指令系统类型	指令	寻址方式	实现方式	其它
CISC（复杂）	数量多，使用频率相差大、可变定长	多种寻址方式	微程序控制技术	周期长，指令直接在主存处理，执行速度慢
RISC（精简）	数量少，使用频率相近，定长格式，多为单周期指令，仅LOAD/Store操作内存	支持方式少	增加通用寄存器；硬布线逻辑控制为主；适合流水线	优化编译要求高，支持高级语言

RISC-V是基于精简指令集原理的开放指令集架构，具有短小精悍、模块化组织的特点，基础指令集仅40多条，可模块化扩展，其ISA可免费使用，允许任何人设计、制造和销售相关芯片与软件，适用于多种应用场景。

IO

工作方式	细分类型	特点与背景解读
程序控制（占用CPU时间最长）	无条件传送	I/O端口始终就绪，CPU可随时直接访问。适用于简单、状态稳定的外设（如LED指示灯、简单按键）。
	程序查询	CPU需持续轮询I/O设备的状态端口，与外设串行工作。适用于低速外设（如早期打印机），但CPU利用率低。
中断	-	CPU发出I/O启动指令后继续执行原任务，外设完成后中断通知CPU。实现CPU与I/O并行，提升CPU效率（如磁盘I/O完成后中断）。
DMA（直接内存存取）	-	由DMA控制器直接完成主存与外设的批量数据传输，仅在传输开始/结束时需CPU干预。用于高速、批量数据传输（如硬盘与内存的大块数据交换）。
通道方式和I/O处理机	-	I/O通道是专用处理机，仅执行I/O操作，无独立内存（与CPU共享）。分为字节多路、数组选择、数组多路通道，进一步解放CPU，适用于大型系统的高并发I/O场景（如大型服务器的多设备并行I/O）。

数据库

数据库（DataBase，DB）定义：是长期存储在计算机内、有组织的、统一管理的相关数据的集合。

- **按存储体系分类**
    - **关系型数据库**：以二维表形式存储数据。
    - **键值（Key - Value）数据库**：将数据存储为键值对集合，键作为唯一标识符。
    - **列存储数据库**：表中数据的存储形式为列。
    - **文档数据库**：可存放并获取文档，支持XML、JSON、BSON等格式。
    - **搜索引擎数据库**：应用在搜索引擎领域的数据存储形式，因搜索引擎会爬取大量数据并以特定格式存储，能保证检索性能最优。

文件（File）：是具有符号名的、在逻辑上具有完整意义的一组相关信息项的集合。

文件系统：是操作系统中实现文件统一管理的一组软件和相关数据的集合，专门负责管理和存取文件信息。

文件的结构和组织

- **文件的逻辑结构**：指文件的组织形式，是从用户角度看到的文件组织形式。用户只需知道文件名就能存取文件信息，无需知道文件的存储位置。分为有结的记录式文件（由一个以上的记录构成）和无结构的流式文件（由一串顺序字符流构成）。
- **文件的物理结构**：从实现的角度看，文件在文件存储器上的存放方式。

- **连续结构（顺序结构）**：将逻辑上连续的文件信息（如记录）依次存放在连续编号的物理块上。只要知道文件的起始物理块号和文件的长度，就可方便地进行文件的存取。
- **链接结构（串联结构）**：把逻辑上连续的文件信息（如记录）存放在不连续的物理块上，每个物理块设有一个指针指向下一个物理块。只要知道文件的第1个物理块号，就可按链指针查找整个文件。
- **索引结构**：将逻辑上连续的文件信息（如记录）存放在不连续的物理块中，系统为每个文件建立一张索引表，索引表记录了文件信息所在的逻辑块号对应的物理块号，并将索引表的起始地址放在与文件对应的文件目录项中。
- **多个物理块的索引表**：索引表在文件创建时由系统自动建立，与文件一起存放在同一文件卷上。根据文件大小不同，索引表占用物理块的个数不等，一般占一个或几个物理块。多个物理块的索引表有链接文件和多重索引方式两种组织方式。

索引分配可增加文件存储容量，包括单级索引、多级索引和混合索引。

索引类型	指向内容说明
直接索引	直接指向存储文件数据的物理盘块（可快速定位小文件数据块）
一级间接索引	指向一个“间接索引块”，该块中存储**多个直接指向数据物理盘块的地址**
二级间接索引	指向一个“二级间接索引块”，该块中存储**多个一级间接索引块的地址**；每个一级间接索引块再指向多个数据盘块地址
三级间接索引	指向一个“三级间接索引块”，该块中存储**多个二级间接索引块的地址**；二级间接索引块→一级间接索引块→数据盘块地址

文件的存取方法：是指读/写文件存储器上一个物理块的方法，通常有顺序存取和随机存取两种。顺序存取是按文件中信息的顺序依次进行读/写；随机存取是对文件中的信息按任意次序随机地读/写。

文件存储空间的管理：常用的空闲空间管理方法有空闲区表、位示图和空闲块链3种。

- **空闲区表**：将外存空间上一个连续的未分配区域称为“空闲区”，适用于连续文件结构。
- **位示图**：在外存上建立一张位示图（Bitmap），记录文件存储器的使用情况，每一位对应文件存储器上的一个物理块，取值0和1分别表示空闲和占用。
- **空闲块链**：每个空闲物理块中有指向下一个空闲物理块的指针，所有空闲物理块构成一个链表，链表头指针放在文件存储器特定位置（如管理块），无需磁盘分配表，节省空间。申请空闲物理块时，根据链表头指针取出第一个空闲物理块，再依指针找后续空闲物理块。
- **成组链接法**：UNIX系统采用的方法，将空闲块分成若干组，每100个空闲块为一组，每组第1个空闲块登记下一组空闲块的物理盘块号和空闲块总数，若某组第1个空闲块号为0，表明是最后一组，无下一组空闲块。

文件共享：不同用户进程使用同一文件，不仅是不同用户完成同一任务的必要功能，还能节省主存空间，减少因文件复制增加的访问外存次数。

- **硬链接**：两个文件目录表目指向同一个索引结点的链接，也称基于**索引结点**的链接，即不同文件名与同一个文件实体的链接。
- **符号链接**：建立新的文件或目录，与原来文件或目录的路径名进行映射，访问符号链接时，系统通过该**映射**找到原文件路径并访问。

文件的保护方式，常采用存取控制的方式，规定不同用户对文件的访问有不同权限，防止文件被未经同意的用户访问，具体方法有：

- **存取控制矩阵**：按个人划分权限，但规模太大。
- **存取控制表**：按用户分类做权限控制。
- **用户权限表**：以用户或用户组为单位，将用户可存取的文件集中存入表中，是对存取控制矩阵的改进。
- **密码**：创建文件时用户提供密码，文件存入磁盘时用该密码对内容加密。

RAID（磁盘阵列）总结

- **定义**：将多个相对便宜的磁盘组合成磁盘组，通过数据分散排列的设计，提升数据安全性和整个磁盘系统的效能。
- **核心优势**：
    - 利用多磁盘提高数据传输率；
    - 通过数据冗余与校验实现可靠性。

RAID等级	类型描述	特点与适用场景
RAID 0	无冗余和无校验的数据分块	具备最高I/O性能和磁盘空间利用率（100%），但无数据保护能力，适用于对性能要求高、非关键数据的场景
RAID 1	磁盘镜像阵列	由磁盘对组成，每个工作盘配有镜像盘（数据完全复制），安全性最高，但磁盘空间利用率仅50%，适合存放重要文件
RAID 2	采用纠错海明码的磁盘阵列	采用海明码纠错技术，需增加校验盘提供单纠错、双验错功能，适合大数据量场景，不适合小数据场景
RAID 3/4	采用奇偶校验码的磁盘阵列	RAID 3采用**位交叉奇偶校验**，RAID 4采用**块交叉奇偶校验**，校验码存于独立校验盘；RAID 3适用于大型文件且I/O需求不频繁的场景，RAID 4适用于大型文件读取场景
RAID 5	奇偶校验码磁盘阵列	无独立校验盘，校验信息分布在组内所有磁盘上，大批量、小批量数据读写性能均较好，磁盘利用率为`(n-1)/n`（`n`为磁盘数）
RAID 6	带双分布式校验的磁盘阵列	设置专用异步校验盘，性能改进有限但成本高昂，磁盘利用率为`(n-2)/n`（`n`为磁盘数）
RAID 10	高可靠性与高性能的组合	基于RAID 0和RAID 1，结合RAID 0的高速读写和RAID 1的数据保护能力，性价比高

中间件

中间件是**分布式系统环境中位于操作系统和应用程序之间的系统级软件**，可在不同技术间共享资源，将不同操作系统、数据库、异构网络环境及若干应用整合成有机协同的工作整体。

1. 是**一类软件**，而非单一软件。
2. 不仅实现**系统互连**，更实现**应用间的互操作**。
3. 基于分布式处理，**网络通信功能**是最突出特点。

支持类型	**核心说明**
交互支持	协调系统中不同组件的通信和数据交换，提供消息队列、远程过程调用（RPC）、对象请求代理（ORB）等机制，实现分布式环境下的进程间通信（IPC），使应用无需关注底层网络细节，专注业务逻辑。
公共服务	提供可复用的服务实现，如事务管理、安全服务、命名和目录服务、持久化服务、负载均衡、故障恢复与容错等，解决分布式系统的一致性、可用性、伸缩性问题。

功能

1. 负责客户机与服务器、客户机与应用层之间的**连接与高效通信机制**。
2. 提供应用层不同服务之间的**互操作机制**，以及应用层与数据库之间的连接和控制机制。
3. 提供多层架构的应用开发、运行平台及开发框架，支持模块化应用开发。
4. **屏蔽硬件、操作系统、网络、数据库的异构差异**。
5. 提供应用的**负载均衡、高可用性、安全机制与管理功能**，以及交易管理机制，保证交易一致性。
6. 提供通用服务，避免重复开发，使应用间可协作。

分类类型	说明及实例
通信处理（消息）中间件	保证不同平台间通信，利用消息传递机制实现分布式系统中可靠、高效、实时的跨平台数据传输；实例：IBM MQSeries。
事务处理（交易）中间件	实现处理顺序协调、监视调度、负载均衡等功能；实例：BEA Tuxedo。
数据存取管理中间件	为不同类型数据的读写、加解密提供统一接口；实例：Windows平台的ODBC、Java平台的JDBC。
Web服务器中间件	提供Web程序执行的运行时容器；实例：Tomcat、JBOSS。
安全中间件	屏蔽操作系统缺陷，提升安全等级；实例：Kerberos、SSL/TLS。
跨平台和架构的中间件	用于开发大型应用软件；实例：CORBA、JavaBeans、COM+模型。
专用平台中间件	为特定应用领域的开发设计提供构件库；实例：Android SDK、iOS SDK。
网络中间件	涵盖网管、接入、网络测试等，是热门研发方向；实例：TCP/IP协议栈、HTTP服务器。

构件

构件（组件）是一个**自包含、可复用的程序集**，可以是源程序或二进制代码的集合。它通过统一的接口对外提供服务，外部只能通过接口与构件交互，不能直接操作构件内部。构件的两个重要特性是**自包含**和**可复用**。

构件是独立的、自包容的，因此架构的开发也是独立的。构件之间通过接口相互协作。

1. **设计构件组装**：规划和设计构件的结构和功能。
2. **建立构件库**：创建和维护可复用的构件库。
3. **构建应用软件**：利用构件库中的构件组装应用程序。
4. **测试与发布**：对组装后的应用进行测试，确保功能正确后发布。

构件组装模型的优点：**易于扩展**：构件的自包含性使系统扩展更容易；**降低成本**：设计良好的构件更容易被重用，降低开发成本；**灵活开发**：构件粒度较小，便于任务分配和团队协作。

构件组装模型的缺点：**对设计要求高**：需要经验丰富的架构师设计构件，不良设计会降低重用性；**可能牺牲性能**：为了提高重用性，可能在性能等方面做出妥协；**学习成本**：要求开发人员熟练掌握构件的使用；**依赖第三方构件质量**：第三方构件库的质量会影响最终软件的质量。

ORB 是 CORBA（Common Object Request Broker Architecture）架构的核心组件，负责实现分布式系统中对象之间的通信和互操作。ORB 的主要特点包括：

1. **通信透明性**：使得应用程序开发者无需关心底层的网络通信细节。
2. **语言和平台无关性**：支持多种编程语言和操作系统平台。
3. **对象引用**：管理对象的引用，使得客户端可以获取和使用远程对象的引用。
4. **服务基础设施**：提供了一系列的服务基础设施，如命名服务、事务服务、安全服务等。
5. **事件通知**：支持事件通知机制，使得应用程序可以接收和处理异步事件。

ORB 的主要功能是实现分布对象系统中的“软总线”，它规定了分布对象的定义（接口）和语言映射，实现对象间的通信和互操作。

CORBA CCM（CORBA Component Model）是 OMG 组织制定的一个用于开发和配置分布式应用的服务器端构件模型规范。它主要包括以下内容：

1. **抽象构件模型**：描述服务器端构件的结构和构件间互操作的结构。
2. **构件容器结构**：提供通用的构件运行和管理环境，并支持对系统服务的集成。
3. **构件的配置和打包规范**：使用打包技术管理构件的二进制代码和配置信息，制定构件包的具体内容和文档内容标准。

嵌入式

嵌入式系统是为了特定应用而设计的专用计算机系统，它紧密结合了信息处理和物理过程。这类系统对功能、可靠性、成本、体积、功耗和环境适应性等综合性能要求非常严格。

分类方式：

按照时间约束：

- **嵌入式实时系统**是指能够在指定或确定时间内完成系统功能，并对外部或内部、同步或异步时间做出响应的系统。其计算的正确性不仅取决于程序的逻辑正确性，还取决于结果产生的时间。如果时间约束得不到满足，系统可能会出错。
    - **强实时（Hard Real-Time）系统**：对时间约束要求非常严格，必须在确定的时间内完成任务。
    - **弱实时（Weak Real-Time）系统**：对时间约束的要求相对较宽松，允许一定程度的时间延迟。
- **嵌入式非实时系统**对时间的约束没有实时系统那么严格，主要关注功能实现而非时间响应。

从安全性的角度，嵌入式系统还可以分为：

- **安全攸关系统（Safety-Critical或Life-Critical）**：其功能的不正确或失效会导致人员伤亡、财产损失等严重后果。
- **非安全攸关系统**：不涉及上述严重后果的系统。

嵌入式系统软件采用层次化结构，具备可配置和可剪裁能力。从现代嵌入式系统的角度来看，嵌入式系统可以分为以下几个层次：

1. **硬件层**：提供运行支撑的硬件环境。包括微处理器、存储器（ROM、SDRAM、Flash等）、I/O接口（A/D、D/A、I/O等）、通用设备以及总线、电源、时钟等。
2. **抽象层**：位于硬件层和操作系统层之间。主要实现对硬件层的硬件抽象（HAL），为上层应用（操作系统）提供虚拟的硬件资源。包括**板级支持包（BSP）**，用于硬件芯片或电路的驱动。
3. **操作系统层**：由嵌入式（实时）操作系统（E(RT)OS）、文件系统、图形用户接口、网络系统和通用组件等可配置模块组成。提供任务管理、存储管理、通信管理、时间管理等功能。
4. **中间件层**：提供数据库（DB）、OpenGL、Java、虚拟机等服务。包括通信中间件如DDS、CORBA、Hadoop等。
5. **应用层**：包括工业控制应用、军事应用、物联网应用、移动设备应用等。面向具体的应用场景，实现特定的功能。

嵌入式软件的主要特点：

1. **可剪裁性**：
    - 嵌入式软件能够根据系统功能需求，通过工具进行适应性功能的加或减。
    - 可以删除系统不需要的软件模块，使系统更加紧凑。
2. **可配置性**：
    - 嵌入式软件能够根据系统运行功能或性能需要进行配置。
    - 能够根据系统的不同状态、容量和流程，对软件工作状况进行扩展、变更和增量服务。
3. **强实时性**：
    - 嵌入式系统中的大多数任务都属于强实时性系统，要求任务必须在规定的时限（Deadline）内处理完成。
    - 软件采用的算法优劣是影响实时性的主要原因。
4. **安全性（Safety）**：
    - 安全性是指系统在规定的条件下和规定的时间内不发生事故的能力。
5. **可靠性**：
    - 可靠性是指系统在规定的条件下和规定的时间周期内程序执行所要求的功能的能力。
6. **高确定性**：
    - 嵌入式软件具有预先设计规划好的行为，其行为随时间、状态的变迁而变化。

安全攸关的软件

美国电气和电子工程师协会（IEEE）将安全攸关软件定义为：“用于一个系统中，可能导致不可接受的风险的软件”。在航空航天、轨道交通和核工业等领域，系统安全性保障至关重要，DO - 178B标准是相关领域保障软件安全性的关键标准，**目标、过程和数据**是该标准的核心，贯穿整个软件生命周期。

目标:依据软件异常导致后果的严重程度，将失效状态分为A到E五个等级，不同等级对应不同的目标数量与简要说明：

等级	失效状态	简要说明	目标数量
A级	灾难性的	软件异常会导致航空器无法安全飞行和着陆	66
B级	危害性的	软件异常会严重降低航空器或机组在克服不利运行情况时的能力	65
C级	严重的	软件异常会显著降低航空器或机组在克服不利运行情况时的能力	56
D级	不严重的	软件异常会轻微降低航空器或机组在克服不利运行情况时的能力	28
E级	没有影响的	软件异常不会影响航空器或机组任何能力	0

过程: DO - 178B标准把软件生命周期分为“软件计划过程”“软件开发过程”和“软件综合过程”。其中，软件开发过程细分为软件需求过程、软件设计过程、软件编码过程和集成过程4个子过程；软件综合过程细分为软件验证过程、软件配置管理过程、软件质量保证过程、审定联络过程4个子过程。

数据: DO - 178B把软件生命周期中产生的**文件、手册、报表、记录**等所有产品统称为软件生命周期数据。

计算机网络

- **计算机网络的功能**：包括数据通信、资源共享、管理集中化、实现分布式处理、负载均衡。
- **网络有关指标（性能指标部分）**：
    - **速率**：指计算机网络上的主机或通信设备在数字信道上传送数据的速率，单位是b/s。
    - **带宽**：有两种含义，一是信号的频带宽度，如传统通信线路上传送电话信号的标准带宽是3.1kHz，单位是赫兹；二是网络通信线路传送数据的能力，即单位时间内从一个结点到另一个结点能通过的最高数据率，单位是b/s。
    - **吞吐量**：表示单位时间内通过某个网络（或信道、接口）的数据量，受网络带宽或网络额定速率限制。
    - **时延**：数据（报文、分组）从网络（链路）一端传送到另一端所需时间，由发送时延（传输时延，从数据块第一个比特开始发送到最后一个比特发送完毕的时间）、传播时延（电或光信号在传输介质传播一定距离的时间）、处理时延（检查分组首部并决定分组导向的时间）、排队时延（分组在队列中等待传输的时间）组成。
    - **往返时间（RTT）**：从发送方发送数据开始，到发送方收到接收方确认（接收方收到数据后立即发送确认）的总时间。
    - **利用率**：有信道利用率（信道被利用的概率，以百分数表示，完全空闲时为零）和网络利用率（全网络信道利用率的加权平均值）两种。
- **非性能指标**：包括费用、质量、标准化、可靠性、可扩展性和可升级性、易管理和维护性。

网络分层和协议

层的名称	主要功能	详细说明
应用层	处理网络应用	直接为端用户服务，提供应用接口和用户接口，例如HTTP、Telnet、FTP、SMTP、NFS等
表示层	数据表示	负责数据编码、格式转换，使应用层能解释数据涵义，例如JPEG、ASCII、GIF、DES、MPEG等
会话层	互连主机通信	管理进程间通信，包括通信控制、检查点设置、重建中断传输链路等，例如RPC、SQL等
传输层	端到端连接	保证数据包无差错、按序、无丢失、无冗余传输，服务访问点为端口，代表性协议有TCP、UDP、SPX等
网络层	分组传输和路由选择	解决路由选择、网络拥塞、异构网络互联问题，服务访问点为逻辑地址（网络地址），代表性协议有IP、IPX等
数据链路层	传送以帧为单位的信息	建立、维持和释放数据链路，分MAC（媒介访问层）和LLC（逻辑链路层）子层，服务访问点为物理地址（MAC地址），代表性协议有IEEE 802.3/2、HDLC、PPP、ATM等
物理层	二进制位传输	定义通信设备的机械、电气、功能、规程特征，代表性协议有RS232、V.35、RJ-45、FDDI等

协议名	要点
FTP	文件传输协议，**数据端口20**，**控制端口21**
TFTP	简单文件传输协议，端口号69
HTTP	超文本传输协议，基于TCP，**端口号80**
SMTP	简单邮件传输协议，用于发送邮件，端口号25
POP3	邮件收取协议，端口号110
Telnet	远程登录协议，**端口号23**
SNMP	简单网络管理协议，端口号161
DNS	域名解析协议，实现域名与IP地址映射，端口号53
TCP	可靠的面向连接传输协议，保证数据有序、无丢失传输
UDP	不可靠的无连接传输协议，适用于对实时性要求高的场景
DHCP	动态主机配置协议，用于自动分配IP等网络参数，端口号67
ICMP	网络控制协议，用于网络故障诊断等（如ping命令基于ICMP）
IGMP	组播协议，用于管理组播组成员关系
ARP	地址解析协议，实现IP地址到MAC地址的映射
RARP	反向地址解析协议，实现MAC地址到IP地址的映射

1. 动态主机配置协议（DHCP）：自动为网络设备分配IP地址、子网掩码、网关、DNS服务器等网络配置信息，简化网络管理。

- **分配方式**：
    - 固定分配：管理员为特定设备指定固定IP。
    - 动态分配：IP地址有租期（如默认8天），到期后需续租或重新分配。
    - 自动分配：长期为设备分配同一IP，类似固定分配的自动化形式。
- **工作过程**：分为DHCP Discover（客户端请求）、DHCP Offer（服务器提供IP）、DHCP Request（客户端确认请求）、DHCP Acknowledge（服务器确认分配）四个阶段。

2. 域名系统（DNS）：将域名解析为IP地址，是互联网的“域名-IP映射地址簿”。

- **解析顺序**：
    - 浏览器端：HOSTS文件→本地DNS缓存→本地DNS服务器→根域名服务器→顶级域名服务器→权限域名服务器。
    - 主域名服务器端：本地缓存记录→区域记录→转发域名服务器→根域名服务器。
- **查询类型**：
    - 迭代查询：服务器返回下一跳查询地址，由客户端自行继续查询。

- 递归查询：服务器直接返回最终IP与域名的映射结果。

3. 传输控制协议（TCP）与用户数据报协议（UDP）

协议	连接性	可靠性	传输效率	适用场景	典型应用
TCP	面向连接	可靠	较低	对可靠性要求高、数据量适中的场景	HTTP、FTP、邮件
UDP	无连接	不可靠	较高	对实时性要求高、可容忍少量丢包的场景	视频直播、游戏、DNS、DHCP、SNMP、TFTP、VOIP

- **TCP特性**：通过三次握手建立连接、四次挥手断开连接，具备流量控制、拥塞控制、数据排序和错误校验机制，保障数据可靠传输。
- **UDP特性**：无连接开销，传输速度快，仅提供简单的错误检测，适用于对延迟敏感的场景。

奇偶校验

在串口通信中，奇偶校验是一种数据校验方法，通信双方需**事先约定使用奇校验或偶校验**。

仅能检测代码中**奇数位出错**的情况，无法发现偶数位出错的场景。

奇校验：若被传输的有效数据中“1”的个数为**奇数个**，校验位填“0”；否则（“1”的个数为偶数），校验位填“1”。

偶校验：若被传输的有效数据中“1”的个数为**偶数个**，校验位填“0”；否则（“1”的个数为奇数），校验位填“1”。

示例

- 奇校验：数据“1000110”中“1”的个数为3（奇数），校验位填“0”，最终传输数据为“1000110（0）”。
- 偶校验：数据“1000110”中“1”的个数为3（奇数），校验位填“1”，最终传输数据为“1000110（1）”。

汉明码

核心思想：在数据中间插入若干校验码，均匀拉大码距。当某一位出错时，会引发多个校验位数值变化，从而实现错误的检测与定位。

海明不等式：`2^r ≥ r + m + 1`

- `r`：校验码的位数；
- `m`：信息位的个数；
- `r+m`：编码后数据的总长度；
- 若满足该不等式，`r`个校验码可判断出具体的出错位。

编码规则：校验码存放在`2^n`的位置（如1、2、4、8等），其余位置为信息位。

CRC校验

广泛应用于网络通信（如以太网、串口通信）及磁盘存储领域，用于检测数据传输或存储过程中的错误。

1. **原理**

通过**多项式（模2除法）运算**实现：将信息位左移k位后，与长度为$k+1$的生成多项式做模2除法，所得余数作为校验字段，最终构成$n+k$位编码（n为信息位长度，k为校验位长度）。

2. **计算步骤（以例题为例：原始信息串10110，生成多项式$G(x)=x^4+x+1$）**
    - **步骤1：信息位后添0**
      生成多项式的阶为r（本题$r=4$），在原始信息位后添加r个0，得到被除数101100000。
    - **步骤2：生成除数**
      多项式$G(x)=x^4+x+1$对应二进制串10011（x^4、x^1、x^0对应位为1，其余位为0）。
    - **步骤3：模2除法运算**
      用被除数101100000除以除数10011，得到余数1111（即CRC校验码）。
    - **步骤4：生成最终信息串**
      将余数1111添加到原始信息10110后，得到最终发送的信息串$10110\ 1111$。
3. **校验过程**
   接收方用相同的生成多项式对收到的信息串做模2除法：
    - 若余数为0，表示数据无错；
    - 若余数非0，则要求发送方重传数据。

网络规划和设计

网络规划设计流程

阶段	核心工作与产物
需求分析	确定业务、用户、应用、计算机平台、网络通信等需求，产物为**需求规范（需求说明书）**
通信规范分析	分析现有网络体系，测量通信量和设备利用率，产物为**通信规范**
逻辑网络设计	生成网络逻辑结构，产物为**逻辑设计文档**（含逻辑设计图、IP地址方案、安全方案等）
物理网络设计	将逻辑设计落地到物理空间，确定网络物理结构，产物为**物理结构设计文档**
实施阶段	完成物理网络的设计、安装与维护

网络分层设计（四层架构）

层级	功能特点
出口层	连接外部ISP（如多运营商网络），实现外部网络接入
核心层	承担高速数据交换、高速传输与出口路由，通过冗余机制保障可靠性
汇聚层	负责网络访问策略控制、数据包过滤、寻址、策略路由及广播域定义等
接入层	实现用户接入、计费管理、MAC地址认证/过滤及用户信息收集

网络冗余设计

- **目的**：避免网络组件单点失效导致应用中断。
- **实现方式**：

- 备用路径：主路径失效时启用，与主路径承担不同网络负载。
- 负载分担：通过并行链路分担流量以提升性能；若存在备用链路，可结合负载分担减轻主路径压力。

IPv6 关键知识点

1. IPv6 优势（对比IPv4）

优势点	说明
地址空间极大	采用128位地址，地址空间较IPv4扩大2^{96}倍
报文头灵活简化	IP报文头部格式更灵活，同时实现了结构简化
支持更多服务类型	对多样化业务场景的支持能力显著增强
协议可演进	允许协议动态新增功能，适配未来技术发展需求
安全性更高	内置身份认证和隐私保护特性，IPsec为强制配置

2. IPv4/IPv6 过渡技术

- 双协议栈技术：节点同时支持IPv4和IPv6协议栈，实现两种协议业务共存。
- 隧道技术：在IPv4网络中部署隧道承载IPv6业务，包括**6to4隧道**、**6over4隧道**、**ISATAP隧道**。
- NAT-PT技术：通过网关设备连接IPv6和IPv4网络，实现协议转换与地址映射。

3. IPv6 地址压缩规则

- IPv6地址共128位，以16位为一段（共8段），每段为4位十六进制数，段间用“:”分隔。
- 压缩规则：
  - 高位0可多次省略；中间段的连续0可用一个0表示；**连续多段0仅能通过“::”代替一次**。

4. IPv6 地址类型

- 三大基础类型：
  - 单播：唯一标识一个IPv6节点的接口，用于点对点通信。
  - 多播（组播）：标识一组IPv6节点的接口，前缀为`11111111`，用于一对多通信。
  - 任意播：指派多个节点的接口，数据报仅传递给其中一个（通常是最近的）接口，前缀固定且其余位为0。
- 单播地址细分：
  - 可聚合全球单播地址：前缀为`001`。
  - 本地单播地址：
    - 链路本地：前缀为`1111111010`。
    - 站点本地：前缀为`1111111011`。

通信原理

- 计算机网络依靠通信技术实现数据在结点间的传送，通信技术是计算机网络的基础。信道分为物理信道和逻辑信道：

    - **物理信道**由传输介质和设备组成，按传输介质不同，可分为有线信道和无线信道。
    - **逻辑信道**是数据发送端和接收端之间的虚拟线路，可分为有连接或无连接的，且以物理信道为载体。
- **香农公式**：用于计算信道容量（信道的最大传输速率），公式为

$$C = B \times \log_2\left(1 + \frac{S}{N}\right) \tag{1}$$

其中

- C 代表信道容量，单位是 b/s；
- B 代表信号带宽，单位是 Hz；
- S 代表信号平均功率，单位是 W；
- N 代表噪声平均功率，单位是 W；
- S/N 代表信噪比（无单位，分贝是信噪比的一种常用表示形式，并非 S/N 本身的单位）。

发信机的信号处理包含信源编码、信道编码、交织、脉冲成形和调制；收信机的信号处理包含解调、采样判决、去交织、信道译码和信源译码:

- **信源编码**：把模拟信号进行模数转换，再做压缩编码（去除冗余信息），最终形成数字信号。像GSM（全球移动通信系统）先通过PCM（脉冲编码调制）编码把模拟语音信号转化为二进制数字码流，再用RPE - LPT（规则脉冲激励 - 长期预测编码）算法压缩。
- **信道编码**：通过增加冗余信息，方便在接收端检错和纠错，解决信道、噪声和干扰引发的误码问题，不过一般只能纠正零星错误，对连续误码没办法。
- **交织**：为解决连续误码导致的信道译码出错问题，将信道编码后的数据顺序按一定规律打乱，接收端在信道译码前再通过交织把数据顺序复原，让连续误码变成零星误码，使信道译码能正确纠错。
- **脉冲成形**：为减小带宽需求，把发送数据转换成合适的波形。因为矩形脉冲要求信道很宽（矩形脉冲竖边垂直需要很高频率），而脉冲成形不要求垂直，所以降低了频率要求。
- **调制**：是将信息承载到满足信号要求的高频载波信号的过程。

复用技术：指在一条信道上同时传输多路数据的技术，常见的有TDM（时分复用）、FDM（频分复用）和CDM（码分复用）等。ADSL使用了FDM技术，语音的上行和下行占用不同带宽。

多址技术：指在一条线上同时传输多个用户数据的技术，接收端可分离多个用户数据，包括TDMA（时分多址）、FDMA（频分多址）和CDMA（码分多址）。

5G通信网络：具有基于OFDM优化的波形和多址接入、实现可扩展的OFDM间隔参数配置、OFDM加窗提高多路传输效率、灵活框架设计、大规模MIMO（Multiple - Input Multiple - Output）、毫米波、频谱共享、先进的信道等特点。

网络分类

网络分类：通常按照覆盖区域和通信介质等特征，可分为局域网（LAN）、无线局域网（WLAN）、城域网（MAN）、广域网（WAN）和移动通信网等。

局域网（LAN）：指在有限地理范围内，将若干计算机通过传输介质互联成的计算机组（通信网络），借助网络软件可实现计算机间文件管理、应用软件共享、打印机共享、工作组内日程安排、电子邮件和传真通信服务等功能。

- **网络拓扑**：局域网专用性强，拓扑结构稳定规范，常见的有星状结构、树状结构、总线结构和环形结构。

- **星型**：中心结点是控制中心，任意两个结点间通信最多两步，传输速度快、网络构简单、建网容易、便于控制和管理，但可靠性低，网络共享能力差，中心结点故障会导致全网瘫痪。
- **树形**：网络成本低，结构简单，结点扩充方便、灵活，方便寻查链路路径，除叶结点及其相连链路外，任何一个工作站或链路故障都会影响整体。
- **总线结构**：各结点设备与一根总线相连，所有结点通过总线传输信息。总线作为数据通信必经之路，负载能力有限（由通信媒体物理性能决定），总线故障会影响总线上每个结点通信。
- **环形结构**：网络中各结点通过首尾相连的通信链路形成闭合环形结构网。各结点地位相同，信息按固定方向单向流动，两节点间仅有一条通路，无信道选择问题，任一结点故障会导致网络物理瘫痪。且环形结构不便于扩充，系统响应延时长，信息传输效率相对较低。
- **网状结构**：网络中任何结点彼此间都存在通信链路，任一结点故障不影响其他结点间通信，但布线繁琐，建设成本高，控制方法复杂。

以太网定义：以太网（Ethernet）是一种计算机局域网组网技术，IEEE制定的IEEE 802.3标准给出了以太网的技术标准。

```
   6 Byte   6 Byte   2 Byte   46-1500 Byte  4 Byte
+--------+--------+--------+---------------+--------+
|  DST   |  SRC   |  Type  |    Payload    |  FCS   |
|  MAC   |  MAC   |        |  (IP/ARP...)  | (CRC)  |
+--------+--------+--------+---------------+--------+
 <-------- Preamble + SFD (7+1 Byte) 物理层添加 ------>

Preamble / SFD : 10101010 10101010 ... 10101011   (物理层，不计入帧长)
DST MAC        : 目的地址
SRC MAC        : 源地址
Type           : 0x0800 IPv4, 0x0806 ARP, 0x86DD IPv6 ...
Payload        : 最小 46 字节，最大 1500 字节 (MTU)
FCS            : 32 位 CRC 校验值
```

以太网结构：最大帧长为1518字节（最大的数据帧为1500字节），**最小帧长为64字节，不足则需加入填充位**。帧头设有32位用于进行CRC32校验，参与校验的是帧头中除前导字段和帧起始符之外的部分。数据帧包含DMAC（目的终端MAC地址）、SMAC（源MAC地址）、Length/Type（长度/类型，2字节，值大于1500代表数据帧类型，小于1500代表数据帧长度）、DATA/PAD（具体数据，若数据长度加帧头不足64字节，需在数据部分增加填充内容）、FCS（帧校验字段，用于判断数据帧是否出错）等字段。当Length/Type取值大于1500时，MAC子层可根据其值直接把数据帧提交给上层协议，由上层协议进行分帧处理，这种结构为较流行的ETHERNET_II协议，大部分计算机都支持。

由于CSMA/CD算法限制，以太网帧的最小长度为64字节。以太网有最大传输距离限制。当通过交换机端口流量过大，超过其处理能力时，会发生端口阻塞。

无线局域网（WLAN）：以无线通信为传输方式的局域网，是实现移动计算机网络的关键技术之一。传输介质：**以微波、激光与红外线等无线电波作为传输介质**，可部分或全部代替传统局域网的有线传输介质。所需设备：架设无线局域网需要无线网卡和访问接入点（AP）。优点：与有线网络相比，具有安装便捷、使用灵活、经济节约、易于扩展等优点。

广域网（WAN）：将分布于更广区域（如一个城市、一个国家甚至国家间）的计算机设备联接起来的网络，由通信子网与资源子网组成。特点：主要提供面向数据通信的服务，支持用户进行远距离信息交换；覆盖范围广，通信距离远，无固定拓扑结构，由电信部门或公司负责组建、管理和维护，向全社会提供有偿服务。

- 通信子网：由通信结点设备和连接这些设备的链路组成。
- 资源子网：主要指网络资源设备，如业务服务器、用户计算机、网络存储系统、网络上运行的各种软件资源、数据资源等。

城域网（MAN）：是在单个城市范围内建立的计算机通信网，覆盖范围介于局域网和广域网之间。其主要技术是DQDB（分布式队列双总线），即IEEE802.6，由**双总线**构成，所有计算机都连接在上面。

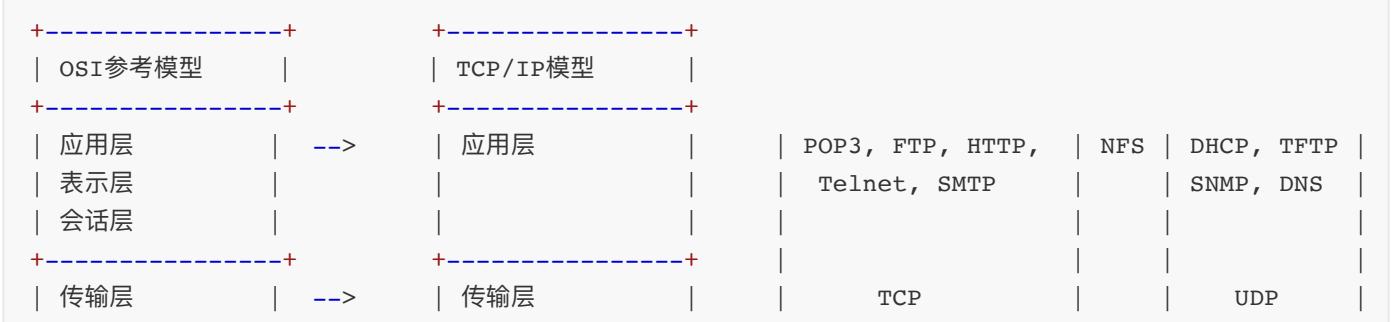
移动通信网：移动通信技术经历了五个发展时期。第一代是模拟通信，采用FDMA（频分多址）调制技术，频谱利用率低；第二代是数字通信系统，采用TDMA（时分多址）数字调制方式，对系统容量限制较大；第三代（3G）采用CDMA（码分多址）数字调制技术，能提供大容量、高质量、综合业务且支持软切换；第四代（4G）包括TD - LTE和FDD

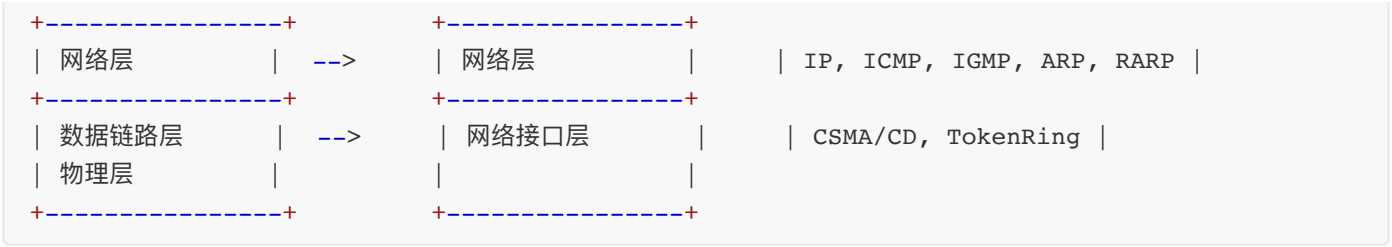
第五代移动通信（5G）：是具有高速率、低时延、大连接特点的新一代宽带移动通信技术，是实现人机物互联的网络基础设施。其三大类应用场景为：**增强移动宽带（eMBB）**，面向移动互联网流量爆炸式增长，提升移动互联网用户应用体验；**超高可靠低时延通信（uRLLC）**，面向工业控制、远程医疗、自动驾驶等对时延和可靠性要求极高的垂直行业应用需求；**海量机器类通信（mMTC）**，面向智慧城市、智能家居、环境监测等以传感和数据采集为目标的应用需求。**网络切片**：5G网络切片可在同一物理网络基础设施上划分为多个逻辑独立的虚拟网络。每个网络切片都是隔离的端到端网络，包含自身独特的延迟、吞吐量、安全性和带宽特性，能灵活应对不同需求和服务。

网络协议和设备

OSI 模型将网络通信从逻辑上分为七层，自顶向下依次是**应用层、表示层、会话层、传输层、网络层、数据链路层、物理层**

- **集线器**：是简单的网络设备，物理层，从一个端口收到的数据会被转送到所有其他端口，还有上联端口用于连接其他集线器或路由设备以形成更大网络。
- **中继器**：工作于OSI体系结构的物理层，接收并识别网络信号后再生信号发送到其他分支，可连接不同物理介质，且需保证每个分支数据包和逻辑链路协议相同。
- **网桥**：工作于OSI体系的**数据链路层**，数据链路层以上各层信息对其透明，包含中继器功能，能连接多种介质和不同物理分支（如以太网、令牌网），使数据包在更大范围传送。
- **交换机**：工作在OSI七层协议中的**数据链路层**，有独立转发通路，能将从一个端口接收的数据经内部处理转发到指定端口，具备自动寻址和交换功能，还可避免端口冲突、提高网络吞吐量。
- 这段内容介绍了计算机网络中的路由器和防火墙两种设备：
- **路由器**：工作在OSI体系结构的网络层，可在多个网络间交换和路由数据包。它通过在相互独立网络中交换路由信息生成路由表，以此进行数据包路径选择，路由表包含网络地址、连接信息、路径信息和发送代价等属性，常用于广域网或广域网与局域网的互连。
- **防火墙**：是网络中重要的安全设备，常作为网络的门户保障网络安全运行。通过设置安全规则对进出网络的数据进行监视和过滤。网络中常采用硬件防火墙，硬件防火墙是将防火墙程序做到芯片里，由硬件执行功能，能减少CPU负担，使路由更稳定，其安全稳定直接关系整个网络的安全。





交换机功能

1. 集线功能：提供大量可供线缆连接的端口，以达成部署星状拓扑网络的目的。
2. 中继功能：在转发帧时重新产生不失真的电信号。
3. 桥接功能：在内置的端口上使用相同的转发和过滤逻辑。
4. 隔离冲突域功能：将部署好的局域网分为多个冲突域，每个冲突域有自己独立的带宽，提升交换机整体带宽利用效率。

基本交换原理

1. 转发路径学习：依据收到数据帧中的源MAC地址，建立该地址同交换机端口的映射，并写入MAC地址表。
2. 数据转发：若交换机根据数据帧中的目的MAC地址，在建立好的MAC地址表中查询到对应条目，就向对应端口进行转发。
3. 数据泛洪：要是数据帧中的目的MAC地址不在MAC地址表中，就向所有端口转发（即泛洪），广播帧和组播帧向所有端口（不包括源端口）进行转发。
4. 链路地址更新：MAC地址表每隔一定时间（如300s）更新一次。

内部网关协议（IGP）：

- RIP - 1、RIP - 2（路由信息协议）：基于距离矢量算法，以跳数为计量标准。
- IGRP（内部网关路由协议）：距离矢量算法，是思科的专有协议。
- EIGRP（增强型IGRP）：拓扑结构变化时才发送路由更新。
- IS - IS（中间系统到中间系统）：基于链路状态路由协议。
- OSPF（开放式最短路径优先）：属于链路状态路由协议，提出了区域（area）的概念。

外部网关协议（EGP）：目前使用的是BGP协议。

计算机网络工程中网络建设工程的三个环节：

- **网络规划**：涵盖网络需求分析、可行性分析，以及对现有网络的分析（当需要对现有网络进行优化升级时）。
- **网络设计**：在网络规划的基础上，设计能解决用户问题的方案，包括确定网络总体目标、总体设计原则，还有通信子网设计、设备选型、网络安全设计等。
- **网络实施**：依据网络设计结果开展设备采购、安装、调试和系统切换（改造升级时）等工作，包含工程实施计划、网络设备验收、设备安装和调试、系统试运行和切换、用户培训等。

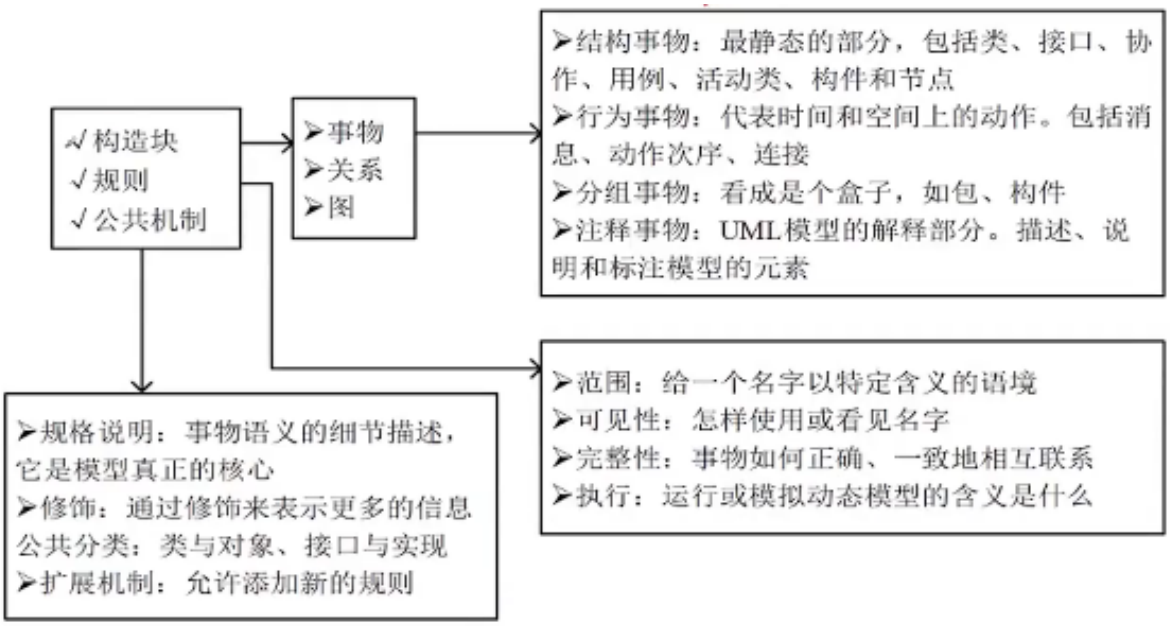
计算机语言

- **机器语言**：用二进制代码表示，计算机能直接识别和执行，由操作码和操作数两部分组成。优点是可被计算机直接理解执行，执行速度快、占用内存少；缺点是难于记忆、编程困难、可读性差，且面向机器，可移植性极差。

- **汇编语言**：用助记符表示各个基本操作的程序设计语言（如用ADD表示加法操作，MOV表示数据传递）。也是面向机器的语言，计算机不能直接执行，通用性和可移植性差，必须经过汇编程序翻译成机器语言程序后才能在计算机上执行，且需要程序员对计算机内部结构非常了解。
- **高级语言**：用接近自然语言和数学语言的语法、符号描述基本操作的程序设计语言。具有简单易学、编程效率高、可移植性好的特点，但需用专门的翻译程序转换成机器语言程序后才能执行。

建模语言

UML 是支持模型化和软件系统开发的图形化语言，是面向对象设计的建模工具，独立于任何具体程序设计语言。



- 构造块
  - 事务：结构事务（静态）、行为事务（动态）、分组事务、注释事务
  - 关系：
    - **依赖**关系（虚线实箭头）
    - **实现**关系（虚线空箭头）
    - **关联**关系（实线）：描述一组链，链是对象之间的连接。包含两种特殊的关系：聚合关系、组合关系
      - **组合**：（实线实菱形，指向整体）强关联、生命周期相同
      - **聚合**：（实线虚菱形，指向整体）弱关联、生命周期可以不同
    - **泛化**：特殊|一般关系（实线空箭头，指向父元素），相对应的，实线实箭头指向子类，则代表**继承**
  - 图（重点：用例图、状态图、活动图）
    - **用例图**：由**参与者（Actor）**、**用例（Use Case）**、**边界以及它们之间的关系**构成，用于描述系统功能，**用例之间的关系有包含、扩展、泛化**。
      - 对系统的语境建模
      - 对系统的需求建模
      - **包含关系**：当两个或多个用例共用一组相同动作时，将这组动作抽为独立子用例供多个基用例共享。因基用例需子用例才完整，子用例必然被执行。在用例图中用带箭头的虚

线表示（标注<>），箭头从基用例指向子用例。

- **扩展关系**：是对基用例的扩展，基用例本身是完整的，无**子用例参与也能完成功能**。在用例图中用带箭头的虚线表示（标注<>），箭头从子用例指向基用例。
- **泛化**：即继承关系的反关系，是特殊/一般的关系，子类继承自父类，父类是子类的泛化。

- **类图**：展现一组**对象、接口、协作**和它们之间的关系，类之间的关系有**关联、依赖、实现、泛化**。
- **对象图**：描述一组对象及它们之间的关系，**是类图中所建立事物实例的静态快照**。
- **构件图（组件图）**：描述了软件的各种构件和它们之间的依赖关系。构件图由源文件代码、二进制代码、可执行文件或动态链接库（DLL）等构件组成，并通过依赖关系相连接。描述一个封装的类和它的接口、端口，以及由内嵌的构件和连接件构成的内部结构。
- **组合结构图**：用于画出**结构化类**的内部内容。
- **顺序图（序列图）**：由一组对象或参与者以及它们之间可能发送的消息构成，是**强调消息时间次序**的交互图。
- **通信图**：强调收发消息的对象或参与者的结构组织，突出**对象之间的组织结构**（关系）。
- **定时图**：强调消息跨越不同对象或参与者的**实际时间**，不只是关心消息的相对顺序。
- **状态图**：描述一个特定对象所有可能的状态，以及因各种事件发生而引起的状态之间的转移和变化。
- **活动图**：将**进程或其他计算的结构**展示为计算内部一步步的控制流和数据流，支持并行。
- **部署图**：展示软件和硬件组件之间的物理关系以及处理节点的组件分布情况。从部署图中，可了解软件和硬件组件之间的物理关系以及处理节点的组件分布情况。能显示运行时系统的结构，同时传达构成应用程序的硬件和软件元素的配置和部署方式。
- **制品图**：描述计算机中一个**系统的物理结构**，通常与部署图一起使用。
- **包图**：描述**由模型本身分解而成的组织单元，以及它们之间的依赖关系**。
- **交互概览图**：是活动图和顺序图的混合物。

- 规则：范围、可见性、完整性、执行
- 公共机制
  - 规格说明：模型的真正和兴
  - 修饰
  - 公共分类
  - 扩展机制

活动图与流程图的区别：

- 描述重点不同：活动图着重表现系统的行为；流程图着重描述处理过程。
- 流程支持不同：流程图一般限于顺序进程；活动图可支持并发进程。
- 设计思想不同：活动图是面向对象的；流程图是面向过程的。

状态图与活动图的区别

建模语言类型	描述	侧重点
状态图	用来描述一个特定的对象所有可能的状态，以及由于各种事件的发生而引起的状态之间的转移和变化	主要描述行为的结果
活动图	将进程或其他计算的结构展示为计算内部一步步的控制流和数据流，主要用来描述系统的动态视图	主要描述行为的动作

顺序图与协作图的区别

建模语言类型	描述	侧重点
序列图	主要用来直观表现各个对象交互的时间顺序，将重点放在以时间为参照，各个对象发送、接收消息，处理消息，返回消息的时间流程顺序，也称为时序图	交互的时间次序
协作图	是一种类图，强调参与交互的各个对象的结构信息和组织	交互的空间结构
共同点	均显示了对象间的交互	-
不同点	序列图强调交互的时间次序；协作图强调交互的空间结构	-

用例图

元素	描述	示例
参与者	系统外部的交互实体，可为用户、外部系统、硬件设备（如IC卡读写器）、时钟（时间/温度触发）等	门禁系统中的IC卡读写器、银行系统的用户
用例	系统中执行的一系列动作，是功能单元	银行的存款、取款、查询余额等功能
通信关联	参与者与用例、用例与用例之间的交互关系	参与者与“存款”用例的关联

用例间的关系

关系类型	描述	示例	图形示意（用例图）
包含（include）	多个用例共用一组相同动作，抽取为子用例；基用例需与子用例结合才完整，子用例必然执行	买饮料、放置饮料、收钱都包含“打开机器”子用例	虚线箭头（<>），基用例指向子用例
扩展（extend）	对基用例的扩展，基用例本身完整；子用例可选执行	读者“还书”用例可扩展“罚款”子用例（逾期时执行）	虚线箭头（<>），子用例指向基用例
泛化	特殊用例是一般用例的子类，体现继承关系的反关系	“注册课程”泛化为“电话注册”和“网上注册”	实线空心三角，子类用例指向父类用例

类之间的关系

关系类型	描述	图形示意
关联关系	类之间的普通联系（如“用户”与“订单”的关联）	实线连接两个类
聚合关系	整体与部分的关系，部分可独立于整体存在（如“班级”与“学生”）	空心菱形+实线，部分类指向整体类
组合关系	整体与部分的关系，部分不能独立于整体存在（如“人体”与“心脏”）	实心菱形+实线，部分类指向整体类
依赖关系	一个类依赖另一个类的存在（如“订单处理”依赖“支付接口”）	虚线+实心三角，依赖类指向被依赖类
泛化关系	子类继承父类的属性和方法（如“动物”与“猫”）	实线+空心三角，子类指向父类
实现关系	类实现接口的方法（如“支付类”实现“支付接口”）	虚线+空心箭头，类指向接口

视图

对于同一个系统，不同人员所关心的内容不同，按照图本身的特点，可将图形划分为用例视图、逻辑视图、进程视图、实现视图和部署视图这5类视图，其中**用例视图居于中心地位**。

视图	要点描述	图形表示	关心人员
用例视图	描述系统的功能需求，系统功能模型	用例图	客户、分析者、设计者、开发者和测试者
逻辑视图	描述如何实现系统内部的功能，系统的静态结构和因发送消息而出现的动态协作关系	类图和对象图、状态图、顺序图、合作图和活动图	系统分析和设计人员
进程视图	描述系统的并发性，并处理这些线程间的通信和同步；将系统分割成并发执行的控制线程及处理这些线程的通信和同步	状态图、顺序图、合作图、活动图、构件图和配置图	开发者和系统集成者
实现视图	描述系统代码构件组织和实现模块及它们之间的依赖关系	构件图	设计者、开发者和测试者
部署视图	定义系统中软硬件的物理体系结构及连接，说明哪个程序或对象驻留在哪个节点上	配置图	开发者、系统集成者和测试者

形式化语言

是把概念、判断、推理转化成特定的形式符号后，对形式符号表达系统进行研究的方法，是用具有精确语义的形式语言书写的程序功能描述，是设计和编制程序的出发点，也是验证程序是否正确的依据。

形式化方法分类：

- 面向对象的形式化方法：通过定义状态和操作进行建模，如 Z 语言、VDM、B、Object - Z 等方法。
- 面向属性的形式化方法：如 OBJ3、Larch 等方法。
- 基于并发性的形式化方法：如 CCS、ACP、CSP、LOTOS 等。
- 基于实时性的形式化方法：如 TRIO、RTOZ 等方法。

Z 语言：是一种*形式化语言*，具有 *“状态 — 操作”* 风格的形式化规格说明语言，在很多大型软件项目中成功应用。其最主要的结构是*模式*，一个模式由*变量说明和谓词约束*两部分组成，可用来描述*系统状态和操作*。Z 语言建立于*集合论和数理逻辑*的基础上，具备将数学进行结构化的方式，能与自然语言结合产生形式化的规格说明。

多媒体

媒体类型	描述	示例
感觉媒体	用户接触信息的感觉形式，直接作用于人的感官，产生视、听、嗅、味、触觉等感觉的媒体	视觉、听觉、触觉
表示媒体	信息的表示形式，感觉媒体转换成表示媒体后可在计算机上加工处理和传输	图像、声音、视频
表现媒体（显示媒体）	表现和获取信息的物理设备，分为输入和输出两类	输入：键盘、鼠标、扫描仪、话筒、数码相机、摄像机；输出：显示器、打印机、音箱、投影仪
存储媒体	用于存储表示媒体的物理介质	硬盘、软盘、光盘、ROM及RAM等
传输媒体	传输表示媒体（即数据流）的物理介质	电缆、光缆、电磁波等

媒体是承载信息的载体，即信息的表现形式，如文字、声音、图像、动画、视频等。多媒体概述中，媒体被分为感觉媒体、表示媒体、表现媒体、存储媒体和传输媒体这五类，从不同维度对承载和传递信息的载体进行了划分。

视音频编码：编解码器是能对信号或数据流进行变换的设备或程序，视音频编码的目的是对视音频数据进行传输和存储。常见的视频文件格式有.mpg、.avi、.mov、.mp4、.rm、.ogg和.tta等。

视音频压缩方法：

- 无损压缩：解压缩后的数据和压缩前完全一致，多数采用RLE行程编码算法，常见格式有WAV、PCM、TTA、FLAC、AU、APE、TAK和WavPack（WV）等。
- 有损压缩：解压缩后的数据与压缩前不一致，压缩过程中丢失一些人眼和耳不敏感的图像或音频信息且不可恢复，常见格式有MP3、WMA、Ogg Vorbis（OGG）等。

数据压缩分为以下3类：

- **即时压缩和非即时压缩**：区别在于信息在传输过程中被压缩还是压缩后再传输。即时压缩一般应用于影像、声音数据的传送；非即时压缩常用到专门的硬件设备，如压缩卡等。
- **数据压缩和文件压缩**：数据压缩专指具有时间性的数据，这些数据常常即时采集、处理或传输；文件压缩是对将要保存在磁盘等物理介质的数据进行压缩。
- **无损压缩与有损压缩**：（此前内容已有介绍，此处可结合理解，无损压缩解压后数据与原数据一致，有损压缩解压后数据与原数据不一致且丢失信息不可恢复）

虚拟现实(VR)/增强现实(AR)技术

- **虚拟现实(VR)**：又称人工现实、临境等，是可以创建和体验虚拟世界的计算机仿真系统，利用计算机生成模拟环境，使用户沉浸其中。其概念包含3层含义：
    - 虚拟实体是计算机生成的逼真实体。
    - 用户可通过自然技能（头部转动、眼动、手势或其他身体动作）与环境交互。
    - 需借助三维传感设备完成交互动作，常用的有头盔立体显示器、数据手套、数据服装和三维鼠标等。
- **增强现实(AR)技术**：把原本在现实世界中难以体验的实体信息（视觉、声音、味道、触觉等），通过模拟仿真后叠加到现实世界中被人类感官感知，从而达到超越现实的感官体验，与多媒体、三维建模、实时跟踪注册、智能交互、传感等技术的发展密切相关。

系统工程

系统工程是为了最好地实现系统目的，对系统的**组成要素、组织结构、信息流、控制机构**等进行分析研究的科学方法，运用各种**组织管理技术**，使系统整体与局部关系协调配合，实现总体最优运行；也是从**整体**出发，合理开发、设计、实施和运用系统科学的工程技术，综合应用自然科学和社会科学的思想、理论和方法，利用电子计算机工具，对系统结构、要素、信息和反馈等进行分析，以达到**最优规划、最优设计、最优管理和最优控制**的目的；还是从**系统观念**出发，以最优化方法求得**系统整体最优**的、综合化的组织、管理、技术和方法的总称。

系统工程方法是一种现代的科学决策方法。

具有整体性、综合性、协调性、科学性和实践性。

包括整体观念、综合观念、科学观念和创新观念等。

系统工程方法	**描述要点**
霍尔的三维结构	由**时间维**、**逻辑维**和**知识维**构成。 时间维：系统工程活动按时间顺序排列的全过程，分为**规划（调研）**、**拟订方案（提出具体计划方案）**、**研制（完成研制方案及生产计划）**、**生产（生产零部件及提出安装计划）**、**安装（安装完毕，完成系统运行计划）**、**运行（系统按照预期开展服务）**、**更新（改进或消亡原有系统）**7个阶段。 逻辑维：时间维每个阶段内的工作内容和思维程序，包括*明确问题*、*确定目标*、*系统综合*、*系统分析*、*优化*、*决策*、*实施*7个步骤。 知识维：专业科学知识。 应用于组织和管理大型工程建设项目。
切克兰德方法	P.切克兰德将其称为“软科学”方法论，**核心是“比较”与“探寻”而非最优化**。 工作过程分为7个步骤：认识问题、根底定义、建立概念模型、比较及探寻、选择、设计与实施、评估与反馈。
并行工程方法	对产品及其相关过程（**制造过程**和**支持过程**）进行并行、集成化处理的系统方法和综合技术。 强调：在产品设计开发期间以最快速度按要求质量完成；各项工作由相关项目小组完成，协调解决问题；依据适当的信息系统工具。
综合集成法	把系统分为**简单系统**和**巨系统**两大类。 开放的复杂巨系统的基本原则与一般系统论原则一致：整体论原则、相互联系的原则、有序性原则、动态原则。
WSR系统方法	是**物理（Wuli）-事理（Shili）-人理（Renli）**方法论的简称，实践准则是“懂物理、明事理、通人理”。 工作过程包括七步：理解意图、制定目标、调查分析、构造策略、选择方案、协调关系和实现构想，协调关系贯穿整个过程。

系统工程生命周期阶段：

- **探索性研究阶段**：识别利益攸关者的需求，探索创意和技术。
- **概念阶段**：细化利益攸关者的需求，探索可行概念，提出有望实现的解决方案。
- **开发阶段**：细化系统需求，创建解决方案的描述，构建系统，验证并确认系统。包括详细计划、开发和验证与确认（V&V）活动。可完全自主选择开发模型，不局限于瀑布或其他计划驱动的方法。

- **生产阶段**：生产系统并进行检验和验证。
- **使用阶段**：运行系统以满足用户需求。
- **保障阶段**：提供持续的系统能力。
- **退役阶段**：存储、归档或退出系统。

系统工程生命周期方法：

- **计划驱动方法**：系统化方法，遵循需求、设计、构建、测试、部署的范式。
- **渐进迭代式开发（IID）**：允许为项目提供初始能力，随之提供连续交付以达到期望的系统，目标在于快速产生价值并提供快速响应能力。适用于需求不清晰不确定或客户希望引入新技术的场景，基于最初假设开发候选系统，评估后若不满足则启动新一轮演进，直到满足利益攸关者要求，适用于较小的、不太复杂的系统。
- **精益开发**：聚焦于向客户交付最大价值并使浪费活动最小化，是动态的、知识驱动的、以客户为中心的过程，使企业人员不断消除浪费以创造价值。
- **敏捷开发**：关键目标在于灵活性。

基于模型的系统工程

- **定义**：基于模型的系统工程（MBSE）是建模方法的形式化应用，以使建模方法支持系统需求、分析、设计、验证和确认等活动，这些活动从概念性设计阶段开始，持续贯穿到设计开发以及后来的所有生命周期阶段。
- **系统工程过程各阶段图形产出**：
    - 需求分析阶段：产生需求图、用例图及包图。
    - 功能分析与分配阶段：产生顺序图、活动图及状态机（State Machine）图。
    - 设计综合阶段：产生模块定义图、内部块图及参数图等。
- **三大支柱**：建模语言、建模工具和建模思路。

系统性能

系统性能评估方法

方法	描述要点
时钟频率法	计算机的时钟频率在一定程度上反映机器速度，同机型时钟频率越高，工作速度越快
指令执行速度法	用加法指令的运算速度衡量计算机速度，单位是MIPS
等效指令速度法	考虑指令比例不同，通过各类指令在程序中所占比例计算计算机运算速度，也称为吉普森或混合比例计算法
数据处理速率法（PDR）	采用计算PDR值衡量机器性能，PDR值越大性能越好。PDR与每条指令和每个操作数的平均位数及每条指令的平均运算速度有关，主要对CPU和主存储器速度进行度量，不适合衡量机器整体速度，未涉及Cache、多功能部件等技术对性能的影响
综合理论性能法	先计算处理部件每个计算单元的有效计算率，再按不同字长调整得出计算单元的理论性能，所有计算单元的理论性能之和即为最终计算机性能
基准程序法	经典评估方法主要针对CPU（有时包括主存）性能，未考虑I/O结构、操作系统、编译程序效率等对系统性能的影响，难以准确评估计算机系统实际性能。把应用程序中用得最多、最频繁的核心程序作为评价计算机性能的标准程序（基准测试程序，benchmark），是目前一致承认的测试系统性能的较好方法

基准测试程序

- 评价程序准确程度递减顺序：**真实的程序>核心程序>小型基准程序>合成基准程序**
- 定义：把应用程序中用得最多、最频繁的核心程序作为评价计算机性能的标准程序，称为基准测试程序（benchmark）
- 类型：整数测试程序、浮点测试程序、Whetstone基准测试程序、SPEC基准测试程序和TPC基准程序

Web服务器性能评估

- 性能指标：**最大并发连接数**、**响应延迟**和**吞吐量**等
- 评测方法：**基准性能测试**、**压力测试**和**可靠性测试**

系统监视方法

- 通过系统本身提供的**命令**，如UNIX/Linux中的W、ps、last，Windows中的netstat等
- 通过系统记录文件，查阅系统在特定时间的行为
- 集成命令、文件记录和可视化技术，如**Windows的Perfmon应用程序**

典型系统的性能关注点

硬件类型	性能指标
计算机	**时钟频率（主频）**、**运算速度**、**运算精度**、内存的存储容量、存储器的存取周期、数据处理速率、吞吐量、各种响应时间、各种利用率、RASIS特性（即可靠性、可用性、可维护性、完整性和安全性）、平均故障响应时间、兼容性、可扩充性和性能价格比
网络	设备级性能指标、网络级性能指标、应用级性能指标、用户级性能指标和**吞吐量**
操作系统	系统上下文切换、**系统响应时间**、**系统的吞吐率（量）**、**系统资源利用率**、可靠性和可移植性
数据库管理系统	数据库的大小、数据库中**表的数量**、单个表的大小、表中允许的记录（行）数量、单个记录（行）的大小、**表上所允许的索引数量**、数据库所允许的索引数量、**最大并发事务处理能力**、负载均衡能力、**最大连接数**等
WEB服务器	**最大并发连接数**、**响应延迟**和**吞吐量**
路由器	**设备吞吐量**、**端口吞吐量**、全双工线速转发能力、背靠背帧数、路由表能力、背板能力、**丢包率**、**时延**、**时延抖动**、VPN支持能力、内部时钟精度、队列管理机制、端口硬件队列数、etc
交换机	交换机类型、配置、**支持的网络类型**、**最大ATM端口数**、etc

计算系统性能的方法

- **定义法**：根据其定义直接获取其理想数据。
- **公式法**：适用于根据基本定义所衍生出的复合性能指标的计算。
- **程序检测法**：通过程序进行实际的测试来得到其实际值（由于测试的环境和条件不定，其结果也可能相差比较大）。
- **仪器检测法**：通过硬件仪器进行测试得到其实际值。

性能调整

- **性能调整的定义**：当系统性能降到最基本的水平时，性能调整由查找和消除瓶颈组成。
- **数据库系统性能调整要点**：主要包括CPU/内存使用状况、优化数据库设计、优化数据库管理以及进程/线程状态、硬盘剩余空间、日志文件大小等。
- **应用系统性能调整要点**：主要包括应用系统的可用性、响应时间、并发用户数以及特定应用的系统资源占用等。
- **性能调整前的准备工作**：必须做的准备工作有识别约束、指定负载、设置性能目标。
- **性能调整的过程**：在建立了性能调整的边界和期望值后，开始一系列重复的、受控的性能试验，循环的调整过程为收集、分析、配置和测试。

阿姆达尔定律（阿姆达尔解决方案）

- **核心思想**：系统中对某部件采用某种更快的执行方式，所获得的系统性能的改变程度，取决于这种方式被使用的频率，或所占总执行时间的比例。
- **加速比公式**：
  加速比 = 不使用增强部件时完成整个任务的时间 / 使用增强部件时完成整个任务的时间
- **题目解析**：
  已知某功能处理时间为系统运行时间的60%（即该部件被使用的比例$F = 0.6$），该功能处理速度提高至原来的5倍（即加速比$S = 5$）。
  根据阿姆达尔定律，整体系统加速比为：
  加速比 $= \frac{1}{(1-F)+\frac{F}{S}} = \frac{1}{(1-0.6)+\frac{0.6}{5}} = \frac{1}{0.4+0.12} = \frac{1}{0.52} \approx 1.923$
  所以答案选 **B**。

信息系统

基本知识

- **概念**：信息系统是由计算机硬件、网络和通信设备、计算机软件、信息资源、*信息用户*和*规章制度*组成的**人机一体化系统**，以处理信息流为目的，具备输入、存储、处理、输出、控制5大基本功能。
- **生命周期**：
  - 产生阶段：需求分析与初步构想。
  - 开发阶段：包含*总体规划*（明确开发目标、架构、实施计划等）、*系统分析*（构建逻辑模型，分析业务流程、数据流程）、*系统设计*（设计物理模型，如架构、数据库、功能模块）、*系统实施*（用户参与的具体实现）、*系统验收*（试运行后验收）。
  - 运行阶段：系统投入使用后的日常运维。
  - 消亡阶段：系统因技术落后或业务变更等退役。

信息系统建设原则

建设原则	要点解读
高层管理人员介入原则	需高层（如CIO）参与，保障战略一致性与资源支持。
用户参与开发原则	贯彻“用户至上”，明确用户范围，核心用户深度参与开发过程，确保系统贴合业务需求。
自顶向下规划原则	以总体规划为指导，在创造性设计中保证信息的一致性，避免局部优化导致整体混乱。
工程化原则	引入“软件工程”思想，规范开发流程、文档与质量管控。
其他原则	创新性（体现先进性）、整体性（保证系统完整）、发展性（具备超前性）、经济性（兼顾实用与成本）。

信息系统开发方法

开发方法	特点与适用场景
结构化法	自顶向下、逐步求精，强调模块化、文档规范化，适用于需求明确、业务流程稳定的项目。
原型法	针对需求不明确的场景，快速构建系统原型并与用户反复迭代；可分为*水平原型*（侧重界面）、*垂直原型*（侧重复杂算法），或*抛弃型*（仅用于需求验证）、*演化型*（逐步迭代成最终系统）。
面向对象法	自底向上抽象“对象-类-类库”，符合人类思维习惯，*复用性强*，适用于复杂业务系统的长期迭代。
面向服务的方法	以“服务”为核心，将相关对象按业务功能分组形成*粗粒度*、*松耦合*的构件，强调接口与实现分离，适用于企业级分布式系统或需要跨系统集成的场景。

业务处理系统TPS

业务处理系统（Transaction Processing System，TPS），又称电子数据处理系统（Electronic Data Processing System，EDP），是计算机在管理领域早期应用的**最初级信息系统形式**。

服务层级与目标：服务于组织管理的**作业层（最低层）**，解决**结构化程度很高**的管理问题，核心目的是帮助作业层管理人员减轻原始数据处理负担，提升具体事务的处理效率。

处理方式与流程：

- 处理方式：支持**批处理**（集中处理大量周期性事务，如月度工资核算）和**联机实时处理**（实时响应事务请求，如银行柜台存款）。
- 流程：涵盖**数据输入→业务处理→文件和数据库处理→文件/报表生成→查询处理**等环节，实现事务数据的全生命周期管理。

背景与意义：作为管理领域信息系统的“基石”，TPS为后续更复杂的信息系统（如管理信息系统MIS、决策支持系统DSS）提供了数据处理能力的底层支撑，是企业数字化管理的早期实践形态。

管理信息系统（MIS）

管理信息系统（Manage Information System，MIS）由**业务处理系统（TPS）**发展而来，在TPS基础上引入大量**管理方法**，对企业整体信息进行处理，利用信息实现预测、控制、计划等功能，辅助企业全面管理。

- **系统组成**：由四大部件构成，分别是
    - 信息源：提供原始数据的来源（如业务单据、外部市场数据）。
    - 信息处理器：负责数据的采集、存储、加工、传输等处理环节。
    - 信息用户：使用系统输出信息的企业各层级管理者（如中层部门经理）。
    - 信息管理者：负责系统规划、开发、维护的人员（如系统管理员、IT部门）。
- **结构类型**：

结构类型	决策过程特点	典型示例
开环结构	执行决策时不收集外部信息，决策不动态调整，事后评价仅为后续决策参考	批处理系统（如月度财务报表批量生成）
闭环结构	决策过程中持续收集信息，不断调整决策	计算机实时处理系统（如库存实时监控与补货决策）

- **价值与意义**：MIS是企业从“事务处理”向“管理决策辅助”升级的关键系统，为中层管理提供结构化信息支持，是后续决策支持系统（DSS）、高管信息系统（EIS）等更高级信息系统的基础，推动企业管理从“经验驱动”向“数据驱动”转型。

决策支持系统（DSS）

- **组成**：由**语言系统**（人机交互接口）、**知识系统**（含数据、模型、规则等知识资源）、**问题处理系统**（整合知识解决决策问题）三个互相关联部分组成的计算机系统。
- **核心特征**：
  - 以**数据和模型**为主要资源，通过定量分析辅助决策。
  - 定位是**支援用户决策**，而非替代用户做决策，尊重决策者的主观判断。
  - 聚焦**半结构化、非结构化问题**（如市场策略优化、风险评估等无固定流程的决策场景）。
  - 目标是**提高决策有效性**（决策质量），而非单纯提升决策效率。
- **结构形式**：
  - 两库结构：由**数据库子系统**（提供数据支持）和**模型库子系统**（提供决策模型，如预测模型、优化模型）组成，通过对话子系统与用户交互（如上图结构）。
  - 基于知识的结构：引入知识管理模块（如专家系统、数据挖掘算法），增强对复杂知识型决策的支持。
- **特点**：
  - 直接**面向决策者**，贴合管理者的决策思维与流程。
  - 重点支持**半结构化问题决策**，填补了TPS（处理结构化事务）、MIS（处理结构化管理信息）的能力空白。
  - 以**辅助、支持决策者**为核心，强调人机协同。
  - 体现决策过程的**动态性**，可根据决策进展实时调整分析逻辑。
  - 提倡**交互式处理**，用户可通过对话系统灵活调整决策参数、选择分析模型。
- **组成（补充）**：包含数据重组（整合多源数据）、知识系统（如数据挖掘技术、分类/聚类智能算法）、模型建立（构建决策分析模型）等环节，为决策提供全方位技术支撑。

专家系统（ES）

是一种智能计算机程序，通过运用知识与推理过程，解决需要资深专家专门知识才能处理的高难度问题。**核心逻辑**：专家系统以“知识库+推理机”为核心，通过知识获取持续丰富领域知识，借助推理机模拟人类专家的推理过程，最终在特定领域实现智能决策与问题解决。

- **核心区别与特征**：

1. **人工智能范畴**：属于人工智能领域，聚焦半结构化或非结构化问题，依赖启发法（如经验性规则）而非传统程序的算法逻辑。
2. **模拟人类推理**：传统应用程序通过数学模型模拟问题，而专家系统模拟人类专家的推理过程。
3. **三要素组成**：由*综合数据库*（描述问题状态）、*知识库*（存储启发式经验知识）、*推理机*（对知识库知识进行推理）构成，对应数据级、知识库级、控制级三级知识结构；传统程序仅含数据和程序两级结构，灵活性不足。
4. **解决实际问题**：处理的是现实中需大量人类专家专门知识的实际问题，而非纯学术问题。
5. **领域局限性**：性能依赖特定领域的人类专家知识，问题领域相对狭窄，通用性较差。

专家系统与一般计算机系统的区别

系统维度	专家系统（ES）	一般计算机系统
功能范围	解决问题 + 解释结果 + **判断与决策**	仅解决问题
处理能力	处理数字与**符号（支持知识推理）**	仅处理数字
问题类型	准结构化/**非结构化问题，可处理不确定知识，聚焦特定领域**	结构化问题，处理确定知识

专家系统的组成及作用

1. **知识库**：存储领域专家的经验知识与专业规则，是系统“智能”的知识基础。
2. **综合数据库**：记录问题初始状态、中间推理结果、用户交互信息等，为推理过程提供数据支撑。
3. **推理机**：控制程序模块，依据知识库的知识进行搜索、推理，是系统的“决策引擎”（如规则解释、逻辑推导）。
4. **知识获取**：具备知识编辑、求精和自学习功能，持续优化知识库，提升系统的知识迭代能力。
5. **解释程序**：向用户解释决策的推理过程，增强系统的透明性与可信度。
6. **人机接口**：分为用户接口（实现“咨询-建议”交互）和专家/知识工程师接口（用于知识录入与更新），是系统与外界的交互桥梁。

办公自动化系统（OA）

以先进科学技术为基础，借助办公自动化设备协助办公人员管理办公信息，核心目标是**提升办公效率与质量**。

- **功能**：
    1. 事务处理：完成日常办公的流程化事务（如请假审批、文件流转）。
    2. 信息管理：由中层管理人员主导，实现信息流的全周期控制（包括信息的收集、加工、传递、存取、分析等）。
    3. 辅助决策：为企业高层及专业智囊团提供信息支撑，辅助其做出行动决策（如战略规划、资源调配）。
- **组成**：
    1. 计算机设备：包含主机系统、终端设备及外部设备（如打印机、扫描仪），是OA系统的硬件计算核心。
    2. 办公设备：涵盖电话机、传真机、复印机、电子会议支持设备（如投影仪、闭路电视）等，满足传统与数字化办公的硬件交互需求。

3. 数据通信及网络设备：保障办公信息的远程传输与共享（如路由器、交换机、VPN设备）。
4. 软件系统：包括系统软件（如操作系统）、专用软件（如OA流程管理系统）、支持软件（如数据库管理工具），为OA系统提供功能实现与运行支撑。

企业资源规划（ERP）详解

ERP 建立在信息技术基础上，融合现代企业先进管理思想，**全面集成企业的物流、资金流、信息流**，为企业提供决策、计划、控制与经营业绩评估的全方位系统化管理平台。它不仅是信息系统，更是一种管理理论和思想。

结构原理（生产管理维度）

层级	作用与内容
生产预测	对市场需求进行预测，为后续计划提供方向，强调计划性。
生产计划大纲	基于经营计划细化生产目标，描述企业在可用资源条件下的产量计划。
主生产计划	进一步细化生产计划大纲，明确“生产什么、生产多少、何时交货”。
物料需求计划	基于主生产计划，规划所需全部制造件和采购件的网络支持与时间进度。
能力需求计划	核算物料需求计划所需的资源能力（如设备、人力），确保生产可行性。
车间作业计划	将零部件生产计划以订单形式下达至对应车间，落实生产执行。

核心功能

1. **支持决策功能**：通过整合的全链路信息（如生产、财务、库存数据），为企业决策提供数据支撑与分析依据，助力战略与运营决策。
2. **行业针对性 IT 解决方案**：针对不同行业（如制造业、服务业）的业务特性，提供定制化的信息系统方案，适配行业需求。
3. **供应链扩展能力**：从企业内部供应链，延伸至全行业、跨行业的供应链管理，实现资源的广泛整合与协同。

电子政务主体与模式分类

电子政务的行为主体主要涉及**政府（Government）、企（事）业单位（Business）、居民（Citizen）**，衍生出以下核心模式：

模式类型	解释与场景
政府对政府（G2G）	政府上下级、不同地区/职能部门间的电子政务活动，例如人口信息共享、跨部门计划管理、财务管理系统、决策支持系统等。
政府对企业/企业对政府（G2B/B2G）	G2B：政府向企业颁发营业执照、许可证、质量认证等； B2G：企业向政府缴税、参与政府项目竞标、提交建议或申诉等。
政府对居民/居民对政府（G2C/C2G）	G2C：政府提供公共安全信息（如社区治安、灾害预警）、户口及证件管理等服务； C2G：居民向政府缴税、反馈民意、使用报警/医疗急救等公共服务。

这些模式通过数字化手段整合政务流程，实现政府内部协同、政企高效互动、政民便捷服务，是政府信息化的核心落地形式。

企业信息化与电子商务核心要点

企业运用信息技术挖掘、编码知识，对业务流程进行管理。

- **实施方向**：
    - 自上而下：需与企业的制度创新、组织创新、管理创新深度结合。
    - 自下而上：以业务人员直接受益和使用水平逐步提高为基础。

企业信息化的目的

类型	具体内容
技术创新	借助信息技术（如计算机辅助设计系统）、互联网掌握创新技术信息，加快技术向生产转化；通过生产技术与信息技术融合，提升技术水平与产品竞争力。
管理创新	整合现有管理流程，从财务、资金管理延伸至技术、物资、人力资源管理，进一步拓展到客户关系管理、供应链管理乃至电子商务领域。
制度创新	创新不适应企业信息化的管理体制、机制和规章制度。

企业信息化规划与数据模型

- **规划基础**：必须建立在企业战略规划之上，以企业战略规划为依据构建管理模式和战略数据模型。
- **战略数据模型分类**：
    - 数据库模型：用于事务级操作，支撑日常业务运转。
    - 数据仓库模型：面向决策分析，为企业战略决策提供数据支持。

企业信息化方法详解

方法类型	核心内容与作用
业务流程重构方法	以“彻底的、根本性的重新设计”为核心思想，打破传统业务流程的束缚，通过重构流程为企业信息化奠定高效的业务运转基础。
核心业务应用方法	强调以企业自身的核心业务为驱动力推进信息化，因为核心业务是企业竞争力的关键，围绕核心业务的信息化能直接提升企业价值。
信息系统建设方法	明确建设信息系统是企业信息化的重点与关键，信息系统是信息化的载体，支撑企业业务处理、数据管理与决策分析等全流程需求。
主题数据库方法	构建面向企业核心业务的数据库，通过统一、共享的主题数据消除“信息孤岛”，实现企业数据的集中管理与高效利用。
资源管理方法	借助计算机技术和网络技术（如ERP、SCM系统），强化企业对人、财、物等资源的整合与管理能力，提升资源利用效率。
人力资本投资方法	将优秀员工视为“资本”，通过培训、发展等投资提升其信息化能力与业务素养，因为人力资本是企业信息化成功的关键保障之一。

信息系统补充

系统需求层次

层次类型	核心内容
战略需求	以提升组织竞争能力、为可持续发展提供支持环境为目标，是企业信息化竞争的基础。
运作需求	是信息化需求的关键一环，包含实现信息化战略目标、运作策略、人才培养三方面内容。
技术需求	因系统开发周期等问题，对系统的完善、升级、集成和整合提出信息技术层面的需求。

系统规划阶段

阶段	核心特征	方法举例
第一阶段	以数据处理为核心，围绕职能部门需求规划	企业系统规划法、关键成功因素法、战略集合转化法
第二阶段	以企业内部管理信息系统为核心，围绕企业整体需求规划	战略数据规划法、信息工程法、战略栅格法
第三阶段	综合企业内外环境，以集成为核心，围绕企业战略需求规划	价值链分析法、战略一致性模型

系统规划方法特点

方法名称	核心特点
企业系统规划法（BSP）	适用于大型信息系统开发，采用“自上而下规划、自下而上实现”，通过**定义管理目标、功能、数据分类、信息结构**（借助CU矩阵）推进。
关键成功因素法（CSF）	识别关键成功因素，提炼**关键信息集合**，确定系统开发优先次序，流程为“目标→识别CSF→确定信息需求（性能指标、数据字典）”。
战略集合转化法（SST）	将企业战略（使命、目标、战略等属性）转化为信息系统的**战略集合**，实现战略与系统的对齐。
战略数据规划法（SDP）	自顶向下全局规划、自底向上详细设计，以**主题数据库**为核心特征（面向业务、信息共享、一次输入、由基本表组成），区分四类数据环境（文件、应用数据库、主题数据库、信息检索系统）。
信息工程方法	以企业内部管理信息系统为核心，将开发过程分为**规划、分析、设计、构建**四阶段，聚焦“信息、技术、过程”三要素。
战略栅格法	通过2×2矩阵（栅格表），从战略影响维度标注现有及未来信息系统组合的特征，评估其对企业生存前景的影响。
价值链分析法	把企业视为“输入-转换-输出”的活动序列，识别**增值/减值**环节，助力企业强化竞争地位。
战略一致性模型（SAM）	划分企业战略与信息化战略的“内外关系”：外部关注竞争环境（如产品、IT市场），内部关注组织结构、信息架构、业务流程等。

供应链管理（SCM）

企业通过改善上下游供应链关系，整合并优化供应链中的信息流、物流和资金流，以获得竞争优势。

- **核心价值**：整合优化供应商、制造商、零售商的业务效率，使商品以正确的数量、品质，在正确的地点、时间，以最佳成本进行生产和销售。
- **基本内容**：涵盖计划、采购、制造、配送、退货五大环节。
- **信息化三流**：
    - 信息流：分为需求信息流（如客户订单、生产计划、采购合同）和供应信息流（如入库单、库存记录、提货发运单）。
    - 资金流：供应链各环节的资金往来与成本控制。
    - 物流：商品的仓储、运输、配送等实体流动。

客户关系管理（CRM）

通过整合人力资源、业务流程与专业技术，为企业客户领域提供集成方案，帮助企业低成本、高效率满足客户需求，建立基于学习性关系的一对一营销模式，提升客户满意度与忠诚度。

- **系统模块**：
    - 销售自动化：最基础模块，实现销售流程的自动化管理（如线索跟进、订单管理）。
    - 营销自动化：支持营销活动的策划、执行与效果分析（如邮件营销、客户分群）。
    - 客户服务与支持：处理客户咨询、投诉，提供售后支持，提升服务体验。
    - 商业智能：通过数据分析挖掘客户需求，辅助营销与服务决策。

数据库、数据仓库、数据湖

维度	数据库	数据仓库
数据组织	按应用组织数据，服务于单一应用	面向主题（如“销售分析”“客户行为”主题）
应用关联	一个应用对应一个数据库	集成多数据源，服务于企业级分析需求
操作类型	支持事务性增删改查	以查询、分析为主，数据相对稳定
价值定位	解决当下业务操作问题	反映历史数据变化，支撑趋势分析、决策支持

维度	数据仓库	数据湖
数据形态	ETL处理后的结构化数据为主	原始数据（含结构化、半结构化数据）
模式定义	存储前定义数据模式，集成前需大量准备工作，数据价值提前明确	存储后定义数据模式，简单集成，数据价值待挖掘
访问方式	标准SQL接口，支持传统BI工具	依赖应用程序或类SQL程序，适配大数据分析框架（如Spark）
核心特性	多源数据集成、数据干净安全、一次转换多次使用	数据“按原样”存储、支持并行计算、性价比高

数据仓库通过**抽取、清理、装载、刷新**等步骤，将多源数据整合入仓库，再通过**OLAP服务器**提供服务，支撑**查询工具、报表工具、分析工具、数据挖掘工具**等应用；也可衍生**数据集市**（面向部门级分析的小型数据仓库），进一步适配细分场景的分析需求。其核心价值是为企业提供历史数据的集中分析能力，助力决策与业务优化。

企业应用集成（EAI）详解

企业应用集成（EAI）适用于实施电子商务的企业及企业间的应用集成，核心是消除“信息孤岛”，实现系统间的协同。其主要集成类型如下：

集成类型	特点与示例
表示集成（界面集成）	**大官网**，最原始的**黑盒集成**，将多个信息系统的界面整合，提供统一入口（如桌面集成），用户看到的是统一界面，底层仍由多系统支撑。
数据集成	**白盒集成**，将不同来源、格式、性质的数据在逻辑或物理上集中，实现全企业数据共享，典型如ETL、数据仓库、联邦数据库。
控制集成（功能集成、应用集成）	业务逻辑层的**黑盒集成**，通过远程过程调用、面向消息的中间件等技术，绑定多个应用系统的功能，实现功能叠加（如钉钉整合多系统功能）。
业务流程集成	最彻底的综合集成，由基于标准、统一数据格式的工作流组成，超越数据和系统层面，实现端到端的流程协同。
门户集成	将企业内部系统对接至互联网，支持外部用户（如客户、合作伙伴）通过门户访问内部资源。

企业应用集成（按数据交换方式分类）

数据交换方式	特点与适用场景
共享数据库	多个应用系统直接共享数据库实现数据交换，**实时性强、支持频繁交互（同步方式）**；但安全性、并发控制、死锁等问题突出，适用于对实时性要求高的内部系统集成。
消息传递	以“消息”（软件对象交互的数结构）为载体，**独立于软件平台，适用于数据量小、要求频繁/立即/可靠/异步交换的场景**（如系统间实时通知）。
文件传输	直接传送数据文件，目标系统读取文件实现数据交换，**可一次性传送大量信息**；但不适合频繁传输，适用于**数据量大、交换频度小、即时性要求低**的场景（如月度财务数据批量同步）。

数据挖掘算法详解

数据挖掘是从海量数据中提取知识的过程，核心功能包括分类、聚类、关联规则、离群点分析等，以下是关键算法的分类与特点：

算法类型	核心功能与典型方法
分类算法	构建区分数据类的模型，预测未知对象的类别。 典型方法：决策树（ID3、C4.5）、K最近邻、贝叶斯、人工神经网络、支持向量机（SVM）。
聚类算法	按“物以类聚”的原则将相似观测值分组。 典型方法：K-means（将数据划分为K个簇，使簇内相似度高）、Apriori（注：Apriori实际为关联规则算法，此处聚类代表为K-means）。
序列模式分析	分析数据间的前后因果关系，挖掘时间或序列上的关联模式（如用户行为的先后逻辑）。
关联分析	挖掘数据间隐藏的相互关系（如“购买尿布的客户常买啤酒”的购物篮分析）。 典型算法：Apriori。
离群点分析	异常检测，识别与大部分对象差异显著的个体（如信用卡欺诈检测）。
回归分析	确定变量间的定量依赖关系（如销售额与广告投入的线性关系）。
决策树	通过构建树状结构分析数据，直观呈现分类/决策逻辑（如ID3基于信息增益、C4.5基于信息增益率选择节点）。
人工神经网络	模拟生物神经网络，具备统计判别、回归、聚类等功能，适用于复杂非线性关系的建模（如图像识别、预测分析）。
遗传算法	模拟生物进化过程，通过“繁殖（选择）、交叉、变异”三个基本过程优化解空间，适用于复杂优化问题（如路径规划、参数寻优）。
关联规则挖掘	挖掘数据间的关联规则（如“购买A商品的客户同时购买B商品”）。 典型算法：Apriori（通过频繁项集推导关联规则）。

企业网站与企业门户分类详解

类型	核心特点与作用
企业网站	侧重消息的**单向传送展示**，缺乏信息互动，是企业门户的雏形。
企业信息门户（EIP）	在互联网模式下，整合**企业的应用系统、数据资源和互联网资源**，统一门户提供产品与用户信息，实现信息共享，助力供应链、客户快速了解企业文化。
企业知识门户（EKP）	在企业网站基础上增加**知识性内容**，员工可了解企业动态、获取项目资源信息，构建企业知识库，提升员工工作效率。
企业应用门户（EAP）	以**业务流程和企业应用**为核心，集成业务流程中功能各异的应用模块，员工和合作伙伴可通过门户访问应用系统，实现移动办公。
企业通用门户	有机融合上述四种门户的功能，提供综合化的门户服务。

电子商务按交易对象分类详解

类型	定义与示例
企业对消费者（B2C）& 消费者对企业（C2B）	B2C：企业面向个人消费者开展的电子商务（如电商平台零售）；C2B：个人向企业提供服务或需求（如IT行业独立咨询师为企业提供咨询服务）。
企业对企业（B2B）	企业之间的电子商务交易（如供应商与制造商的原材料采购、企业间的服务协作）。
消费者对消费者（C2C）	个人之间的电子商务交易（如二手物品交易平台、个人技能服务平台）。
企业对政府（B2G）	企业与政府之间的电子商务活动（如政府采购企业产品、企业参与政府项目竞标）。

信息安全基础知识

信息安全概念详解

信息安全属性

属性	定义
机密性	确保信息不暴露给未授权的实体或进程。
完整性	只有得到允许的人才能修改数据，且能判别数据是否被篡改。
可用性	授权实体在需要时可访问数据，攻击者不能占用资源阻碍授权者工作。
可控性	可控制授权范围内的信息流向及行为方式。
可审查性	对信息安全问题提供调查的依据和手段。

信息安全的范围

范围类型	核心内容
设备安全	是信息系统安全的物质基础，包括三个方面： ① *设备稳定性*：设备在一定时间内不出故障的概率； ② *设备可靠性*：设备在一定时间内正常执行任务的概率； ③ *设备可用性*：设备可以正常使用的概率。
数据安全	采取措施确保数据免受未授权的泄露、篡改和毁坏，包括三个方面： ① 数据*秘密性*：数据不受未授权者知晓的属性； ② 数据*完整性*：数据正确、真实、未被篡改、完整无缺的属性； ③ 数据*可用性*：数据可以随时正常使用的属性。
内容安全	是信息安全在政治、法律、道德层次的要求，包括三个方面： ① 信息内容在*政治上健康*； ② 信息内容*符合国家法律法规*； ③ 信息内容*符合中华民族优良道德规范*。
行为安全	确保信息系统行为安全以保障整体安全，包括三个特性： ① 行为*秘密性*：行为过程和结果不危害数据秘密性； ② 行为*完整性*：行为过程和结果不危害数据完整性，且过程和结果符合预期； ③ 行为*可控性*：行为过程可被观测、控制和纠正。

信息存储安全

包括信息使用的安全、系统安全监控、计算机病毒防治、数据加密与防非法攻击等。

1. **用户的标识与验证**：
    - 目的：限制访问系统人员，验证用户身份合法性。
    - 方法：
        - 基于特殊安全物品：如智能IC卡识别法、磁条卡识别法。
        - 基于物理特征：如签名识别法、指纹识别法、语音识别法。
2. **用户存取权限限制**：
    - 目的：限制进入系统的用户操作范围。
    - 方法：
        - 隔离控制法：通过物理隔离、时间隔离、逻辑隔离、密码技术隔离等方式建立屏障。
        - 限制权限法：对用户分类管理，严格控制目录、文件的访问权限，防止越权操作。

计算机病毒特点，具有**隐蔽性、传染性、潜伏性、触发性、破坏性**。

防治措施

1. 及时下载安装安全补丁、升级杀毒软件。
2. 定期检查敏感文件。
3. 使用高强度、不同账号差异化的口令。
4. 每天备份重要数据。
5. 安装公安部认证的防病毒软件，定期全盘病毒检测与清除。

6. 部署防火墙，提高系统安全性。
7. 不使用时断开网络连接，重要系统与互联网物理隔离。
8. 不打开陌生人邮件附件，谨慎处理熟人邮件附件。
9. 正确配置系统并规范使用病毒防治产品。

网络安全

网络威胁表现

1. 非授权访问：如假冒身份、非法用户入侵或合法用户越权操作。
2. 信息泄露或丢失：敏感数据在传输、存储中被窃取或意外丢失。
3. 破坏数据完整性：非法修改、删除、插入数据以干扰正常使用。
4. 拒绝服务攻击：干扰网络服务系统，导致响应缓慢甚至瘫痪。
5. 利用网络传播病毒。

安全措施目标

1. 认证：确保会话对方身份真实一致。
2. 访问控制：确保会话对方操作权限合法。
3. 完整性：确保接收信息与发送信息一致。
4. 审计：确保交易可事后证实（不可抵赖性）。
5. 保密：确保敏感信息不被窃听。

网络安全协议详解

协议名称	核心功能与特点	所在OSI层级
TLS（传输层安全协议）	保障传输层数据传输的安全性，提供加密、完整性验证等能力。	传输层
IPSEC	在IP协议中新增**认证头（AH）**和**封装安全载荷（ESP）**两种密码安全机制： - AH：支持IP数据的认证性和完整性； - ESP：实现通信机密性，对IP包加密。	网络层
PGP协议	基于RSA的邮件加密软件，也可用于文件存储加密；支持PGP证书和X.509证书，证书包含版本号、公钥、持有者信息、数字签名、有效期、对称加密算法偏好等信息。	应用层
HTTPS	使用端口443，核心作用： - 建立信息安全通道，保证数据传输安全； - 确认网站真实性（身份认证）。	应用层

补充说明：SSL（TLS前身）在传输层，防火墙、IPSec在网络层，链路加密、PPTP/L2TP在数据链路层等，体现了网络安全在各层级的防护布局。

信息系统安全系统框架详解

信息安全系统框架由**技术体系、组织机构体系、管理体系**三大模块构建，具体内容如下：

从实现技术维度，涵盖多方面安全技术：

- **基础安全设备**：包括密码芯片、加密卡、身份识别卡等，为安全提供硬件支撑。
- **计算机网络安全**：防范信息在网络传输过程中被未经授权破坏、更改和盗取，保障数据传输安全。
- **操作系统安全**：要求操作系统无错误配置、无漏洞、无后门、无特洛伊木马，防止非法用户对计算机资源的非法存取。
- **数据库安全**：分为数据库管理系统安全和数据库应用系统安全，涉及物理/逻辑数据库完整性、元素安全性、可审计性、访问控制、身份认证、可用性等技术。
- **终端设备安全**：从电信网终端设备角度，包含电话密码机、传真密码机、异步数据密码机等。

组织机构体系是信息系统安全的组织保障，由**机构、岗位、人事机构**三个模块构成，明确安全管理的组织架构与人员职责。

管理体系由**法律管理、制度管理、培训管理**三部分组成，从合规、流程、人员能力层面保障信息系统安全的可持续性。

对称密钥加密算法

维度	内容描述
定义	加密密钥和解密密钥相同，或可从一个推导出另一个，又称**私钥加密算法**。
优点	加密速度快、效率高。
缺点	加密强度不高；密钥分发困难，传输需依赖安全可靠的途径。
应用	适用于**大量数据的加密**场景。
实现方式	分组密码（对明文分组和密文分组进行运算）、序列密码（对明文和密文数据流按位或字节运算）。
常见算法	DES（*56位密钥*、64位数据块）、3DES（*112位密钥*，加密流程：K1加密→K2解密→K1加密；解密流程：K1解密→K2加密→K1解密）、IDEA（128位密钥）、AES、RC-5等。

非对称密钥加密算法

维度	内容描述
定义	加密密钥和解密密钥完全不同，分为**公钥**（公开）和**私钥**（私有），且无法从一个推导出另一个，又称**公钥加密算法**。
优点	加密强度高；密钥分发简单；适应开放性使用环境；可实现数字签名与验证。
缺点	加密速度慢；算法复杂。
应用	适用于**少量数据的加密**场合。
常见算法	*RSA（密钥长度512位*，安全性依赖于“大数分解为两个素数”的难度，主要用于数字签名）、Elgamal（基于有限域离散对数难题）、ECC（椭圆曲线算法）、Diffie-Hellman等。

数字摘要

维度	内容描述
定义	又称**杂凑算法**或**单向散列函数**，利用哈希（Hash）函数对数据加密：输入任意长度字符串，输出定长的“消息摘要”（Hash值/散列值）。
核心特性	单向性（无法从摘要反推明文）、抗弱碰撞性（难找到另一明文生成相同摘要）、抗强碰撞性、定长输出（与明文长度无关）。
典型算法	- MD5：将任意长度字节串转换为定长大数，确保信息传输完整一致，*散列值为128位*。 - SHA：*散列值为160位*，安全性高于MD5；国密算法SM3，散列值为256位。
注意事项	数字摘要仅保证**数据完整性**，不保证机密性（明文“裸奔”），因此常与加密算法结合使用。

数字证书

又称数字标识，由认证中心（CA）签发，是标识网络用户身份的电子文档，用于认证用户公钥，遵循国际X.509体系标准。

- **内容**：版本信息；唯一的证书序列号；签名算法；发行机构名称（采用X.500格式）；有效期；所有者名称；所有者公钥信息；发行者对证书的数字签名。

密钥管理技术（PKI）

是CA安全认证体系的基础，为密钥管理提供平台，属于网络安全技术和规范。

- **组成**：由公钥证书、证书管理机构、证书管理系统、相关软硬件设备及法律基础共同组成，其中**公钥证书**是

核心部分。

- **核心角色**：
  - **CA（认证中心）**：负责签发证书、管理证书、撤销证书。
  - **RA（注册登记证书机构）**：接收证书申请人的注册信息，审批证书申请、恢复密钥申请、恢复证书申请。
- **流程示例**：用户向RA提交证书申请→RA确认用户身份→RA将申请提交给CA→CA签发证书并将公钥登记录入证书目录→CA将证书传给RA→RA将证书传给用户。

密钥管理技术详解

对称密钥的分配与管理

- **密钥分配需解决的问题**：
  - 引入自动分配密钥机制，提升系统效率。
  - 尽可能减少系统中驻留的密钥量。
- **对称密钥的分配方式（用户A与B的4种方式）**：
  1. A选取密钥，通过物理手段发送给B。
  2. 第三方选取密钥，通过物理手段分别发送给A和B。
  3. A、B事先已有一个密钥，一方选取新密钥后，用已有密钥加密该新密钥并发送给另一方。
  4. 三方A、B、C各有保密信道，C选取密钥后，分别通过A、B各自的保密信道发送。

公钥（非对称密钥）加密体制的密钥管理

管理方式	核心内容与特点
公开发布	用户将自己的公钥发送给其他用户或向某一团体广播，方法简单但**易被伪造**（如假冒用户发送公钥）。
公用目录表	由可信实体维护公钥动态目录，管理员为每个用户建立含用户名和公钥的条目；用户需安全注册公钥。 **缺点**：若攻击者获取管理员密钥，可伪造公钥目录表，假冒用户并监听消息。
公钥管理机构	由专门机构维护公钥动态目录，采取更严密的控制措施；用户通过机构的公钥加密通信（机构用私钥加密应答，用户用其公钥解密）。 **缺点**：管理机构易成为系统瓶颈，且公钥目录表易被篡改。
公钥证书	由证书管理机构（CA）签发，包含用户公钥、身份、时间戳等信息，CA用私钥签名。 用户交换证书并通过CA的公钥验证，避免了统一机构管理的不便与安全隐患。

访问控制技术

- **三要素**：
  - 主体（Subject）：主动实体，如用户、用户组、终端、应用程序等。
  - 客体（Object）：被动实体，如可操作的信息、资源、对象等。

    - 控制策略（KS）：主体对客体的操作行为集和约束条件集，是授权规则的集合。
- **核心内容**：包括认证、控制策略、审计三个方面。

技术类型	核心特点与适用场景
访问控制矩阵	以主体为行索引、客体为列索引的矩阵，每个元素表示访问方式集合。 **缺点**：查找、实现不方便。
访问控制表	每个客体对应一个表，记录有权访问该客体的主体信息。 **优点**：方便查询客体的访问主体；**缺点**：难以查询一个主体对所有客体的访问权限。
能力表	每个主体对应一个表，记录该主体对所有客体的访问权限。 **优点**：方便查询主体的所有访问权限；**缺点**：难以查询对某一客体具有访问权限的主体。
授权关系表	每行表示主体和客体的一个授权关系，对应访问矩阵的非空元素。 **优点**：适合用关系数据库实现；按主体排序可达到能力表效率，按客体排序可达到访问控制表效率。

数字签名

通过单向函数处理报文，生成用于认证报文来源、核实报文是否篡改的字母数字串，基于公钥算法（非对称密钥技术）实现。解决**否认、伪造、篡改、冒充**问题，保障数据的完整性和不可抵赖性。

签名过程

1. 发送者A用散列函数计算消息摘要（MD）。
2. A用自己的私钥（PrA）加密原文（M）和MD，得到`PrA(M+MD)`。
3. 用接收者B的公钥（PB）加密上述信息，得到`PB(PrA(M+MD))`。
4. B用自己的私钥（PrB）解密，得到`PrA(M+MD)`。
5. 用A的公钥（PA）解密，得到M和MD。
6. B计算M的消息摘要，与解密的MD对比，一致则数据未被篡改。

维度	数字加密	数字签名
加密密钥	接收者的公钥	发送者的私钥
解密密钥	接收者的私钥	发送者的公钥
核心作用	保障数据机密性	保障数据完整性、不可抵赖性

签名条件：

1. 签名可信：使接收者相信签名者是慎重签字的。
2. 不可伪造：证明是签字者本人而非他人签字。
3. 不可重用：签名是文件的一部分，不能移到其他文件。
4. 签名的文件不可改变：签名后文件不能再被修改。
5. 不可抵赖：签名者事后不能否认签字行为。

信息安全的抗攻击技术

一、密钥的选择

- **密钥分类**：数据加密密钥（DK，直接加密数据）、密钥加密密钥（KK，保护密钥安全传递）。
- **密钥生成考虑因素**：
    - 增大密钥空间；
    - 选择强钥；
    - 保证密钥的随机性。

二、拒绝服务攻击（DoS）与防御

- **定义**：借助网络系统/协议缺陷，使网络拥塞、资源耗尽或应用死锁，阻碍正常服务响应（常见为分布式拒绝服务攻击DDoS）。
- **攻击分类**：
    - 消耗资源：占用网络连接、磁盘空间、CPU/内存资源，使系统无暇处理正常请求。
    - 破坏或更改配置信息：篡改服务程序、系统/用户启动文件，改变服务提供方式。
    - 物理破坏网络部件：破坏计算机、路由器、电源等关键设备。
    - 利用服务程序处理错误使服务失效。
- **DDoS攻击结构**：采用三级控制结构——Client（攻击者主机，发起/控制攻击）、Handler（主控端，控制代理端）、Agent（代理端，实施实际攻击）。
- **防御方法**：
    - 加强数据包特征识别，定位攻击源；
    - 防火墙监视敏感端口（如UDP 31335、TCP 27665等）；
    - 统计通信数据量，识别异常流量；
    - 及时修复系统漏洞。

三、欺骗攻击与防御

1）ARP欺骗

- **定义**：又称ARP毒化，欺骗局域网内PC的网关MAC地址，导致网络不通，攻击者可截取或篡改数据包。
- **防范措施**：
    - 固化ARP表（如WinXP下执行`arp -s gate-way-ip gate-way-mac`）；
    - 使用ARP服务器；
    - 采用双向绑定方法；
    - 部署ARP防护软件（如ARP Guard）。

（2）DNS欺骗

- **定义**：冒充域名服务器，将查询IP指向攻击者，使用户访问虚假主页。
- **检测方法**：
    - 被动监听检测：捕获DNS请求-应答包，若同一请求对应多个不同应答则可疑；

    - 虚假报文探测：向非DNS服务器发请求，若收到应答则存在攻击；
    - 交叉检查查询：反向查询应答IP对应的DNS名，不一致则被欺骗。

（3）IP欺骗

- **定义**：修改IP数据包源地址，隐藏发送方身份或冒充其他系统，常用于发起DDoS攻击。
- **防御方法**：入口过滤（在网络边缘设备检查IP数据包源标头，拒绝可疑包）。

四、端口扫描

尝试与目标主机端口建立连接，通过端口响应判断是否开放（“活动端口”），用于识别目标主机开放的服务及操作系统。

五、强化TCP/IP堆栈以抵御拒绝服务攻击

- **SYN Flood（同步风暴）**：利用TCP三次握手缺陷，发送大量伪造源地址的SYN报文，使目标主机半连接队列满，拒绝新连接。
- **ICMP攻击**：
    - “Ping of Death”：发送超过64KB的ICMP数据包，导致主机内存分配错误、TCP/IP堆栈崩溃；
    - “ICMP风暴”：大量发送ICMP数据包，消耗主机CPU资源致系统瘫痪。
- **SNMP攻击**：利用TCP/IP网络管理协议（SNMP），控制交换机、路由器等设备，重定向通信流、篡改配置甚至接管网络。

信息系统安全保护登记

级别	名称	要点解读
第1级	用户自主保护级	通过*隔离用户与数据，实现自主访问控制*，为用户提供手段保护自身及用户组信息，防止其他用户非法读写、破坏数据。
第2级	系统审计保护级	实施更细粒度的自主访问控制，通过*登录规程*、*审计安全事件*、*隔离资源*，使用户对自身行为负责。
第3级	安全标记保护级	具备系统审计保护级所有功能，增加*安全策略模型数据标记*、*主体对客体的强制访问控制非形式化描述*，能准确标记输出信息，消除测试发现的错误。
第4级	结构化保护级	扩展第三级的自主和强制访问控制至*所有主体与客体*，*考虑隐蔽通道*，*加强鉴别机制*，*支持系统管理员和操作员职能*，提供可信设施管理，增强配置管理控制，系统具有相当抗渗透能力。
第5级	访问验证保护级	满足访问监控器需求，访问监控器仲裁所有主体对客体的访问且自身抗篡改，设计实现时*最小化复杂性*，支持安全管理员职能，扩充审计机制（安全事件发信号），*提供系统恢复机制*，系统具有很高抗渗透能力。

信息安全风险管理详解

信息安全风险是指各类应用系统及其赖以运行的基础网络、处理的数据和信息，因可能存在的软硬件缺陷、系统集成缺陷，以及信息安全管理中潜在的薄弱环节，而导致的不同程度安全风险。

风险评估实施流程

步骤	说明
确定风险评估的范围	明确需要评估的信息系统、数据或业务范围。
确定风险评估的目标	明确评估要达成的目的（如识别高风险点、支撑安全决策等）。
建立适当的组织结构	组建包含技术、管理、业务人员的评估团队，明确职责分工。
建立系统的风险评估方法	选择或设计适合的风险评估模型、工具和流程。
获得最高管理者对风险评估策划的批准	确保评估工作获得组织高层的认可与资源支持。

风险评估基本要素与内涵

- **基本要素**：*脆弱性*、*资产*、*威胁*、*风险和安全措施*。
- **评估内涵**：对信息资产存在的脆弱性、面临的威胁、造成的影响，及三者综合作用所带来的风险的可能性评

估。

风险计算模型与流程

- **关键要素及属性**：
  - 信息资产：属性为**资产价值**；
  - 弱点/脆弱性：属性为**弱点被威胁利用后对资产影响的严重程度**；
  - 威胁：属性为**威胁发生的可能性**。
- **风险计算过程**：
  1. 识别信息资产并对资产赋值（量化资产重要性）；
  2. 分析威胁并对威胁发生的可能性赋值；
  3. 识别信息资产的脆弱性并对弱点严重程度赋值；
  4. 根据威胁和脆弱性计算安全事件发生的可能性；
  5. 结合信息资产的重要性和安全事件发生的可能性，计算信息资产的风险值。

软件工程

软件工程概述

将系统化的、严格约束的、可量化的方法应用于软件的开发、运行和维护，同时也对这些方法进行研究。

- **软件生命周期**：软件从*需求分析*、*软件设计*、*软件开发*、*运行维护，直至被淘汰*的全过程。
- **软件工程支持活动（PDCA循环）**：
  - **Plan（规划）**：进行软件规格说明，规定软件的功能及其运行时的限制。
  - **Do（执行）**：开展软件开发，开发出满足规格说明的软件。
  - **Check（检查）**：实施软件确认，确认开发的软件能够满足用户的需求。
  - **Action（改进）**：推进软件演进，软件在运行过程中不断改进以满足客户新的需求。
- **软件生命周期模型**：为使软件生命周期中各项任务有序按规程进行，对任务进行规程约束的工作模型，主要开发模型包括瀑布模型、原型化模型、螺旋模型、增量模型、迭代模型、敏捷方法等多种类型。

开发模型-瀑布模型

维度	内容描述
定义与阶段	又称生命周期法，是结构化方法中常用的开发模型，将软件开发过程划分为**软件计划、需求分析、软件设计、程序编码、软件测试、运行维护**6个阶段。
优点	1. 为项目提供按阶段划分的检查点； 2. 前一阶段完成后，仅需关注后续阶段； 3. 提供统一模板，使分析、设计、编码、测试等工作有共同指导。
缺点	1. 各阶段间产生大量文档，增加工作量； 2. 线性开发模式导致用户需到开发末期才能看到成果，增加开发风险； 3. 不适应用户需求变化，且需求分析阶段难以完全获取需求； 4. 前期错误若传到后期可能扩散，引发项目失败。
适用场景	适用于**需求明确且很少变更**的项目。

开发模型-原型化模型

维度	内容描述
定义	在获取一组基本需求定义后，利用高级软件工具可视化开发环境，快速建立目标系统最初版本，交给用户试用、补充和修改，反复迭代直至用户满意（得出系统“精确解”）。
原型分类	- 按是否实现功能：水平原型、垂直原型； - 按最终结果：抛弃型原型（仅用于需求验证，开发后抛弃）、演化型原型（逐步迭代完善，最终成为产品）。
适用场景	适用于**用户需求不明确**的场合。
关键因素	原型法成败及效率高低，取决于**模型的建立及建模的速度**。
阶段划分	1. 原型开发阶段： - 途径一：模拟软件系统的人机界面和人机交互方式； - 途径二：真正开发一个原型； - 途径三：找来一个或几个正在运行的类似软件进行比较； 2. 目标软件开发阶段：基于验证后的原型，开发最终目标软件。

开发模型-螺旋模型

螺旋模型是在**快速原型**基础上扩展而成，融合**瀑布模型的系统性**与**演化模型的迭代性**，核心强调**风险分析**，通过“迭代-风险评估-优化”循环推进软件开发。

核心阶段（四步循环）

阶段	关键动作与目标
目标设定	开展需求分析，明确阶段目标、过程/产品约束，制定详细管理计划
风险分析	识别可选方案的风险，制定解决措施，规避潜在问题
开发与验证	基于风险评估选择开发模型，开展原型开发（如软件产品原型）并验证有效性
评审	评审项目进展，决定是否进入下一轮螺旋循环；若继续，制定下一阶段计划

特点与适用场景

- **特点**：
  - 兼具瀑布模型的“阶段化、系统性”和演化模型的“迭代优化”，通过风险分析降低项目失败概率；
  - 支持**增量式开发**，可随迭代逐步完善需求与功能。
- **适用场景**：
  - 大型、复杂的软件开发项目（如企业级管理系统、嵌入式系统）；
  - 需频繁应对需求变化、技术风险的场景，或多种开发方法（面向规格、面向过程、面向对象）混合的项目。

螺旋模型通过“风险前置分析+迭代验证”，在保障项目可控性的同时，提升了对复杂需求和技术不确定性的适应能力，是大型软件项目管理中平衡“规范性”与“灵活性”的重要方法。

敏捷开发模型

敏捷模型是一种**强调迭代、增量、适应性和以人为本**的软件开发方法，适用于需求变化快、不确定性高的项目，核心特点如下：

- **适应性（Adaptive）而非预设性（Predictive）**：不预先僵化规划全流程，而是通过短周期迭代快速响应需求变化（类比“发射导弹”需动态调整，区别于瀑布模型“发射火箭”的预设性）。
- **面向人（People-oriented）而非面向过程（Process-oriented）**：重视团队协作、开发者自主决策与直接沟通，而非严格遵循流程文档。

主流敏捷方法解析

方法	核心特点与适用场景
极限编程（XP）	以“交流、朴素、反馈、勇气”为价值观，采用小周期迭代，强调**结对编程、测试先行、持续集成**，适合小型高效团队。
水晶方法	提倡“机动性”，根据项目规模、团队特性选择流程（如“水晶黄”适配小型团队，“水晶黑”适配大型团队），灵活性极强。
Scrum	侧重项目管理，通过**迭代（Sprint，1-4周）**推进，核心角色包括产品负责人（管理需求优先级）、敏捷教练（保障流程）、自组织团队，配套每日站会、评审会、回顾会机制。
特征驱动开发（FDD）	以“特征”为核心，分5个过程（开发对象模型、构造特征列表、计划特征开发、特征设计、特征构建），适合大型团队与复杂业务系统。

Scrum是最具代表性的敏捷方法，流程如下：

1. **需求收集**：产品负责人从用户处获取需求，整理为**产品待办列表（Product Backlog）**。
2. **迭代规划**：团队在迭代计划会议中选择本轮迭代（Sprint）需完成的工作，形成**迭代待办（Sprint Backlog）**。
3. **迭代执行**：团队在1-4周内自组织完成工作，每日站会同步进度。
4. **成果交付与复盘**：迭代结束后召开评审会（展示可发布的产品增量）和回顾会（总结改进点），同时整理下一轮待办。

敏捷最佳实践

- **基本原则**：短平快会议、小型版本发布、较少文档、客户直接参与等，是敏捷在流程、协作、文档管理上的具体体现，核心是通过轻量化、高频互动提升开发效率与响应速度。
- **四大价值观**：沟通、简单、反馈、勇气（源于极限编程，是敏捷文化的核心）。
- **五大原则**：快速反馈、简单性假设、逐步修改、提倡更改、优质工作。
- **十二个最佳实践**：计划游戏、小型发布、隐喻、简单设计、测试先行、重构、结对编程、集体代码所有制、持续集成、每周工作40小时、现场客户、编码标准等，从流程、协作、技术层面保障敏捷落地。

类型	具体实践与价值
规划协作	计划游戏（明确需求优先级）、小型发布（快速交付价值）、隐喻（统一团队认知）、现场客户（用户深度参与）
开发质量	简单设计、测试先行、重构、结对编程、集体代码所有制、编码标准
工程效率	持续集成（频繁合并+自动化测试）、每周工作40小时（平衡效率与健康）

构件组装模型（基于构件的软件开发，CBSD）

是利用**模块化方法**，将系统拆分后复用构件库中的软件构件，通过组合高效构造应用软件的过程。融合螺旋模型的演化、迭代特征，本质上是**增量式、复用驱动**的开发方法。

阶段	关键动作
需求分析和定义	明确系统需求，识别可复用构件的需求场景
体系结构设计	设计系统的模块化架构，确定构件间的交互方式
构件库的建立	开发或收集可复用的软件构件（如UI组件、业务逻辑模块），形成构件库管理体系
应用软件构建	从构件库中选取合适构件，通过组合、适配构建应用系统
测试和发布	对集成后的系统进行测试，验证构件兼容性与系统功能，最终发布

- **特点**：强调构件复用，大幅提升开发效率；迭代演化，可快速响应需求变化；依赖构件库的成熟度。
- **适用场景**：有大量可复用构件的领域（如企业级ERP、政务系统）；需求相对稳定但需快速迭代的项目。

V模型

是**瀑布模型的变种**，核心强调**“测试与开发阶段一一对应”**，且“测试计划先行”，通过分阶段测试保障每个开发环节的质量。

阶段对应关系

开发阶段	对应测试阶段	核心目标
需求分析	验收测试	验证系统是否满足用户最终需求
概要设计	系统测试	验证系统整体架构、模块间交互是否符合设计要求
详细设计	集成测试	验证模块集成后的功能、性能是否符合详细设计
编码	单元测试	验证单个代码单元（函数、类）的逻辑正确性

- **特点**：测试阶段明确，与开发阶段强绑定，确保每个环节的产出都经过验证；流程线性，变更成本高。
- **适用场景**：需求明确、变更极少的项目（如军工软件、金融核心系统）；对质量要求极高，需严格阶段把控的场景。

快速应用开发模型（RAD）

快速应用开发（RAD）是**增量型软件开发过程模型**，当项目模块化要求较高时，通过大量复用可复用构件，采用基于构件的建造方法，以极短开发周期完成软件开发。其核心是“**快速迭代+构件复用**”，类似蒙娜丽莎从草稿到成品的快速绘制过程，逐步增量完善。

阶段	关键动作与目标
业务建模	分析业务流程，识别核心业务逻辑、角色交互，明确业务需求边界。
数据建模	设计数据结构、数据库 schema，定义数据的存储、流转和关联规则。
过程建模	结合业务与数据，设计具体操作流程、功能逻辑，明确模块间交互。
应用生成	利用复用构件、快速开发工具（如低代码平台）快速生成应用原型或核心功能。
测试与交付	对应用进行快速测试（侧重功能验证），快速交付并根据反馈迭代优化。

- **特点**：
  - 增量开发，快速迭代；
  - 强调模块化和构件复用，依赖成熟的构件库；
  - 开发周期短，需配合快速开发工具（如RAD工具、低代码平台）。
- **适用场景**：
  - 需求相对明确、模块化程度高的项目（如企业内部审批系统、小型业务管理应用）；
  - 需快速交付的业务场景，追求“短平快”上线并迭代优化。
- **价值**：大幅缩短开发周期，快速验证需求，适合市场响应型项目。
- **局限**：对构件库成熟度、开发工具依赖性强；需求变更频繁或模块化程度低的项目，易出现“构件适配困难”导致效率下降。

能力成熟度模型集成（CMMI）

CMMI 是用于评估和改进软件（及工程领域）组织过程能力的模型，帮助组织从“依赖个人的混乱开发”逐步走向“标准化、量化管理，最终持续优化”的成熟过程。它广泛应用于软件企业的过程改进、资质认证场景，是衡量组织软件工程能力的重要标尺，助力企业提升项目成功率、产品质量与组织效率。

级别	核心特征与意义
初始级	过程无秩序、混乱，项目成功依赖个人努力或机遇，缺乏标准化流程，风险较高。
已管理级	建立基本项目管理过程，类似项目可复制成功，具备基础的计划、跟踪和控制机制。
已定义级	软件过程完全文档化、标准化，形成组织级标准软件过程，过程可复用、可统一改进。
量化管理级	通过详细度量标准量化管理过程，依赖数据进行分析和控制，过程稳定性、可预测性大幅提升。
优化级	持续开展过程改进，利用量化数据和创新方法不断优化，实现过程的长期高效迭代。

需求分类

按需求内容分类

类型	定义与示例
业务需求	客户提出的宏观功能需求，具整体性、全局性。如“搭建一套覆盖全流程的电商管理系统”。
用户需求	设计员调研的每个用户具体需求。如电商系统中“客服需快速查询订单状态并回复客户”。
系统需求	整合后的最终需求，含功能、性能、设计约束等。 - 功能需求：软件必须完成的基本动作（如电商系统“自动生成订单报表”）。 - 性能需求：静态/动态数值要求（如“系统响应速度≤2秒”“每秒处理500笔订单”）。 - 设计约束：受硬件等限制的要求（如“必须兼容现有仓储硬件接口”）。

从客户角度分类

类型	定义与示例
基本需求	明确规定的功能，产品必备。如“手机能通话、发短信”。
期望需求	客户认为理应包含的功能，未明确提出但缺失会不满。如“手机支持高清拍照”。
兴奋需求	客户未要求但具备后能大幅提升满意度的功能。如“手机支持卫星通信紧急呼救”。

需求工程

需求工程由**需求开发**和**需求管理**两部分构成，是软件开发中管理需求的完整体系：

模块	定位与核心动作
需求开发	主线与目标，包含*需求获取*（收集用户/业务需求）、*需求分析*（拆解与验证需求）、形成*需求规*格（文档化需求）、*需求确认与验证*（确保需求准确）。
需求管理	支持与保障，涵盖*需求文档的追踪管理*、*变更控制*（规范需求变更流程）、*版本控制*（管理需求文档的迭代版本）等管理性活动。

需求获取

步骤	核心动作
开发高层业务模型	构建宏观业务逻辑框架，明确业务整体目标与流程
定义项目范围和高层需求	划定项目边界，明确核心高层需求，可借助系统上下文图、顶层用例图等建模手段
识别用户角色和用户代表	梳理参与系统的各类用户角色（如管理员、普通用户），选取有代表性的用户代表参与需求调研
获取具体需求	深入挖掘用户的具体功能、性能等需求
确定目标系统的业务工作流	梳理系统涉及的业务流程，明确各环节的输入输出与交互逻辑
需求整理与总结	对获取的需求进行分类、优先级排序，形成初步需求文档

方法

方法	特点与适用场景
用户面谈	1对1或多对多，了解用户主观想法，交互性好，但成本高，需领域知识支撑，适合深入挖掘个性化需求
问卷调查	适合用户多、地理位置分散的场景，成本低但无法了解细节，用于获取普适性需求
现场观察	针对复杂流程和操作，实地观察用户工作场景，挖掘隐性需求
头脑风暴法	集思广益，适合团队共创需求，整合多方想法
情节串联版	用图片展示用户场景，友好但耗时，适合直观呈现需求逻辑
JPR联合需求计划	高度组织化群体会议，各方参与，交互好但成本高，适合大型项目的需求共识确认
原型法	用于需求不确定的场合，通过可交互原型快速验证需求，减少误解

需求变更管理

变更控制过程

1. 识别问题：发现需求变更的触发点（如用户新诉求、业务变化）。
2. 问题分析和变更描述：分析变更的原因、范围，明确变更后的需求内容。
3. 变更分析和成本计算：评估变更对时间、资源、成本的影响。
4. 变更实现：执行变更并验证修改后的需求。

常见策略

- 所有需求变更必须遵循变更控制过程，确保流程规范。
- 未获批准的变更，不得开展设计和实现工作，避免无效投入。
- 变更由项目变更控制委员会决策，确保客观性。

- 项目风险承担者需了解变更内容，明确影响范围。
- 不得删除或修改变更请求的原始文档，保证可追溯性。
- 每个集成的需求变更需跟踪到经核准的变更请求，保持水平可追踪性。

需求跟踪矩阵

- **正向跟踪**：检查《产品需求规格说明书》中的每个需求是否能在设计文档、代码、测试用例等后续工作成果中找到对应点，确保需求被完整实现。
- **逆向跟踪**：检查设计文档、代码、测试用例等工作成果是否都能在《产品需求规格说明书》中找到出处，避免无需求依据的冗余开发。
- **目的**：建立并维护“需求-设计-编程-测试”之间的一致性，确保所有工作成果符合用户需求。

需求定义方法

方法	特点与适用场景
严格定义法	所有需求能被预先定义，开发与用户可准确交流，可用预先的方法、术语或图形描述，适合需求明确、稳定的项目
原型法	适用于需求不确定的场景，通过原型促进项目参与者交流，需求确定后可遵循严格定义法迭代完善，适合创新型、需求模糊的项目

系统分析 & 系统设计

结构化分析 & 面向对象分析。结构化设计 & 面向对象设计

结构化分析

结构化分析依据**分解与抽象原则**，按照系统数据处理流程，通过**数据流图（DFD）**建立系统功能模型，完成需求分析。

元素	说明	图形示意
数据流	由固定成分数据组成，代表数据流向，以明确定义的名字表示含义。	→
加工	描述输入数据流到输出数据流的变换，即输入数据经处理后生成输出数据流。	○
数据存储	以记录文件或记录表形式存储数据。	——（或矩形）
外部实体	存在于软件系统之外的人员或组织，是数据的发源地与归宿地。	□

- **功能模型**：通过**数据流图（DFD）**体现，聚焦系统数据处理流程。
- **行为模型**：通过**状态转换图（STD）**体现，包含状态（初态、终态）、事件，描述系统行为变化。
- **数据模型**：通过**实体联系图（ER）**体现，包含实体、联系，描述数据间的关系。
- **数据字典**：定义数据元素、数据结构、数据流、数据存储、加工逻辑、外部实体等，是各模型的“数据说明

书”。

界面设计三原则

原则	核心含义与价值
置于用户控制之下	让用户主导界面操作，如提供明确的操作反馈、可撤销/恢复功能等，提升用户掌控感与使用安全感。
减少用户的记忆负担	避免让用户记忆过多信息，如采用直观的布局、明确的标签、历史记录功能等，降低用户认知成本。
保持界面的一致性	统一界面的布局风格、操作逻辑、视觉元素（如按钮样式、颜色含义），提升用户操作习惯的延续性，减少学习成本。

结构化设计（SD）

结构化设计（SD）是**面向数据流**的设计方法，以软件需求规格说明（SRS）和结构化分析（SA）阶段的数据流图、数据字典为基础，采用**自顶向下、逐步求精、模块化**的过程，分为**概要设计**和**详细设计**两个阶段：

- **概要设计**：确定软件系统结构，进行模块划分，明确每个模块的功能、接口及模块间调用关系。
- **详细设计**：为每个模块设计实现细节。

模块设计核心原则：高内聚、低耦合

内聚（模块内部代码联系的紧密程度）

内聚类型	描述
功能内聚	完成单一功能，各部分协同工作、缺一不可，内聚程度最高。
顺序内聚	处理元素相关，且必须顺序执行。
通信内聚	所有处理元素集中在一个数据结构的区域上。
过程内聚	处理元素相关，且必须按特定顺序执行。
时间内聚	所包含的任务必须在同一时间间隔内执行（如初始化模块）。
逻辑内聚	完成逻辑上相关的一组任务（内聚程度较低）。
偶然内聚	完成一组无关联或松散关联的任务（内聚程度最低）。

耦合（模块之间联系的程度）

耦合类型	描述
非直接耦合	两个模块无直接关系，通过上级模块控制和调用实现联系，耦合程度最低。
数据耦合	模块间通过参数传递简单数据，耦合程度较低。
标记耦合	模块间通过参数传递记录等复杂信息（如数据结构）。
控制耦合	模块间传递的信息包含用于控制模块内部逻辑的信息。
通信耦合	模块共用输入信息或输出需整合以形成完整数据（共享输入/输出）。
公共耦合	多个模块访问同一公共数据环境（如全局数据结构、共享通信区）。
内容耦合	一个模块直接访问另一个模块的内部数据；或模块间代码重叠、入口异常等，耦合程度最高。

设计阶段详解:

- 概要设计：核心产出是**系统结构图**，用于展现系统的模块划分、模块功能及模块间的调用关系，实现从数据流图到模块结构的转化。
- 详细设计：需设计每个模块的实现算法和局部数据结构，其表示工具包括：
    - **图形工具**：业务流程图、程序流程图、PAD（问题分析图）、NS流程图等。
    - **表格工具**：通过表格列出所有可能的操作和对应条件，描述过程细节。
    - **语言工具**：用高级语言或伪码（如PDL）描述过程细节。

结构化编程

- **设计方法**：采用**自顶向下、逐步求精**的设计方法，模块通过“顺序、选择、循环”的控制结构连接，且每个模块只有**一个入口和一个出口**。
- **核心原则**：程序 =（算法）+（数据结构），算法和数据结构作为独立整体分开设计，以算法（函数或过程）为主。
- **32字设计准则**：自顶向下，逐步细化；清晰第一，效率第二；书写规范，缩进格式；基本结构，组合而成。

数据库设计（E-R图联系类型）:E-R图用于描述现实世界的概念模型，其中实体间的联系存在三种一般性约束：

联系类型	描述
一对一（1:1）	A中的每个实体在B中至多一个实体对应，反之亦然。
一对多（1:N）	A中的每个实体在B中有多个实体对应，B中每个实体在A中至多一个实体对应。
多对多（M:N）	A中的每个实体在B中有多个实体对应，反之亦然。

面向对象分析（OOA）

- **对象构成世界**：客观世界由对象组成，对象包含**对象名、属性、操作（方法）**，可按属性分类，对象间通过**消息传递**实现联系，具有**封装性、继承性、多态性**三大特性。
- **开发过程**：以用例驱动、体系结构为中心，采用**迭代渐增式**，包含需求分析、系统分析、系统设计、系统实现4个阶段。

OOA模型架构

- **5个层次**：主题层、对象类层、结构层、属性层、服务层。
- **5个活动**：标识对象类、标识结构、定义主题、定义属性、定义服务。
- **对象类结构**：
    - 分类结构：体现“一般与特殊”的关系（如“动物”与“猫”“狗”）。
    - 组装结构：反映“整体与部分”的关系（如“汽车”与“车轮”“发动机”）。

OOA核心原则

原则	核心含义与价值
抽象	舍弃个别、非本质特征，抽取共同、本质特征。包含**过程抽象**（操作序列视为单一实体）和**数据抽象**（数据与操作结合为对象，外部仅知“做什么”，不知“怎么做”，是OOA核心）。
封装	将对象的属性和操作结合为不可分割的系统单位，隐蔽内部细节，保障对象独立性。
继承	特殊类继承一般类的全部属性和操作，使系统模型简练清晰，减少冗余。
多态	同一操作作用于不同对象时，可产生不同解释和执行结果，提升系统灵活性。
分类	将具有相同属性和操作的对象划分为一类，是抽象原则在对象描述中的表现。
聚合（组装）	将复杂事物视为简单事物的组装体，简化对复杂事物的描述（如“电脑”由“CPU”“内存”等组装而成）。
关联	基于事物间的实际联系产生联想，是人类思考问题的常用方法。
消息通信	对象间仅通过消息通信，不允许直接存取内部属性，体现封装性，用于表示对象间动态联系。
粒度控制	面对复杂问题域时，控制视野：考虑全局时关注大组成部分，关注细节时暂时撇开其余部分。
行为分析	分析事物复杂行为及相互依赖、交织的关系，明确对象间的行为逻辑。

OOA实施步骤

步骤	核心动作
确定对象和类	识别问题域中的对象，并将具有相同属性和操作的对象归类。
确定结构	明确对象类之间的分类结构（一般-特殊）和组装结构（整体-部分）。
确定主题	对对象类和结构进行分组，形成高层模块，简化系统复杂度。
确定属性	定义对象的属性（数据特征），明确其取值和约束。
确定方法	定义对象的操作（行为特征），明确其功能和实现逻辑。

软件测试

动态测试分类

类型	核心特点与方法
黑盒测试	不考虑程序内部结构，仅在接口处测试；用例设计方法：等价类划分、边界值分析、错误推测、因果图。
白盒测试	基于程序内部结构（又称结构测试）；用例设计方法：基本路径测试、循环覆盖测试、逻辑覆盖测试（语句、判定、条件等覆盖）。
灰盒测试	介于黑盒与白盒之间，兼顾接口与部分内部逻辑的测试。

测试阶段划分

1. 单元测试：模块级测试，针对软件最小单元（模块）。开发人员负责，采用白盒测试方法，借助**驱动模块**（模拟主程序）和**桩模块**（模拟子模块）。计划：在软件详细设计阶段完成。
2. 集成测试：组装测试，验证模块间协作性。策略：一次性组装、增量式组装（自顶向下、自底向上、混合式）。执行：以黑盒测试方法为主，计划在软件概要设计阶段完成；自顶向下组装无需驱动模块，自底向上组装无需桩模块。
3. 确认测试（有效性测试）定位：验证软件功能、性能是否符合用户需求。计划：在需求分析阶段完成。类型：
    1. 内部确认测试：开发组织内部按需求说明书测试。
    2. Alpha测试：用户在开发环境下测试。
    3. Beta测试：用户在实际使用环境下测试。
    4. 验收测试：交付前以用户为主的测试。
4. 系统测试：定位：结合软件、硬件、外设、网络等元素，验证整个系统是否满足需求规格。内容：功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、安装与反安装测试。计划：在系统分析阶段（需求分析阶段）完成。

逻辑覆盖方法（白盒测试核心技术）

覆盖类型	核心要求
语句覆盖	每条语句至少执行一次。
判定覆盖	每个判定的每个分支至少执行一次。
条件覆盖	每个判定的每个条件取到所有可能的值。
判定/条件覆盖	同时满足判定覆盖和条件覆盖。
条件组合覆盖	每个判定中各条件的所有组合至少出现一次。
路径覆盖	程序中每条可能的路径至少执行一次。

其他测试类型

类型	核心定义
压力测试	确定系统瓶颈或性能上限，获取最大服务级别。
负载测试	验证系统在超负载环境下的承担能力。
强度测试	系统资源极低时，考查软件运行情况（测下限）。
容量测试	确定系统可处理的同时在线最大用户数。
AB测试	Web/App多版本对比，收集用户体验和业务数据以选最优版本。
Web测试	针对Web应用的测试，包含链接测试、表单测试等。
自动化测试	将人工测试行为转化为机器自动执行，提升测试效率。

净室软件工程（CSE）

是一种应用**数学与统计学理论**，以经济方式生产高质量软件的工程技术，目标是通过严格工程化过程实现开发中“零缺陷”或接近零缺陷。强调在**规约和设计阶段消除错误**，而非事后修复；使用**盒结构规约**进行分析和设计建模，以**正确性验证（而非测试）**为主要纠错机制，通过统计测试获取软件可靠性（出错率）信息。

理论	核心内容
函数理论	程序可视为“定义域（所有输入序列）到值域（对应输出集合）”的映射。函数需满足： - 完备性：每个输入必有输出； - 一致性：每个输入仅对应一个输出； - 正确性：可通过函数理论推理验证程序设计的正确性。
抽样理论	因无法测试所有软件使用场景，将所有可能的使用情况视为“总体”，通过统计学抽样测试样本，分析软件性能与可靠性。

手段	核心说明
统计过程控制下的增量式开发	将开发划分为一系列小增量，成员每次仅关注部分工作，降低整体复杂度。
基于函数的规范与设计（盒子结构）	定义三个抽象层次： - 黑盒（行为视图）：刻画系统行为，通过“激发-反应”规则描述系统对事件的响应； - 状态盒（有限状态机视图）：以类似对象的方式封装状态数据和操作，描述输入（激发）与输出（反应）； - 清晰盒（过程视图）：定义状态盒的变迁功能，即过程设计。
正确性验证	净室软件工程的核心，通过严格推理验证设计正确性，大幅提升软件质量。
统计测试和软件认证	采用统计学抽样原理，通过“使用模型”生成测试用例（样本），推导系统预期操作性能的统计结果。

缺点

1. 理论性强，需深厚数学知识，正确性验证步骤困难且耗时；工程师需特殊训练，开发成本高昂。
2. 不进行传统模块测试，与实际开发场景存在脱节（如工程师对开发环境不熟悉、编译器/操作系统Bug可能引发未预期错误）。
3. 脱胎于传统软件工程，不可避免带有其部分弊端。

软件项目管理

软件项目管理是为使软件项目按预定**成本**、**进度**、**质量**顺利完成，对**人员（People）**、**产品（Product）**、**过程（Process）**、**项目（Project）** 进行分析和管理的活动。

进度管理（时间管理）

工作分解结构（WBS）：将项目按原则分解为任务，任务再分解为工作，最终分配到日常活动，即“项目→任务→工作→日常活动”。

工作分解基本要求（原则）：

原则	说明
工作包可控可管理	WBS的工作包不能过于复杂，需具备可控性与可管理性。
分解层级适度	任务分解不能过细，WBS树形结构一般不超过6层。
交付成果明确	每个工作包需有一个明确的交付成果。
完成标准清晰	每个任务必须有明确定义的完成标准。
利于责任分配	WBS设计需支持清晰的责任分配机制。

任务活动图

- **活动定义**：确定完成项目交付成果所需的具体活动，明确每个活动的**前驱关系**、**持续时间**、**必须完成日期**、**里程碑或交付成果**。
- **表示方法**：
    - 前导图法（单代号网络图法）
    - 箭线图法（双代号网络图法）

进度管理主要活动过程

1. 活动定义
2. 活动排序
3. 活动资源估算
4. 活动历时估算
5. 制定进度计划
6. 进度控制

软件配置管理（SCM）

软件配置管理（Software Configuration Management, SCM）是**标识、组织和控制修改的技术**，应用于整个软件工程过程。其目标是**标识变更、控制变更、确保变更正确实现并向相关人员报告变更**，以最小化错误并最有效地提高生产效率。

核心内容

- **版本控制**：管理软件配置项的版本演化。
- **变更控制**：并非阻止变更，而是**对变更进行管理，确保其有序进行**。引发变更的因素包括：
  - 外部变更要求：如客户需求、工作范围修改（**最难处理**，因IT项目需求变更概率大、后期工作量增幅显著）。
  - 内部变更要求：如开发过程中为修复测试错误而修改源码、设计。
- **权限管理**：所有配置项的操作权限由**CMO（配置管理官员）**严格管理：
  - 基线配置项：向开发人员开放**读取权限**。
  - 非基线配置项：向PM（项目经理）、CCB（变更控制委员会）及相关人员开放。
- **状态与版本号**：

状态	说明	版本号格式
草稿	配置项刚建立时的状态，可修改；通过评审后转为“正式”。	0.YZ（YZ：01~99，递增）
正式	配置项通过评审后的状态，是基线的一部分；若需修改，转为“修改”状态。	X.Y（X：1~9主版本号；Y：0~9次版本号，首次正式为1.0）
修改	正式配置项被修改时的状态；修改完毕并重新通过评审后，转回“正式”。	X.YZ（修改时增大Z值；修改完成后Z置0，X.Y递增）

配置库分类

配置库类型	别名/说明	可修改性
动态库	开发库、程序员库、工作库；保存开发人员正在开发的配置实体。	可随意修改
受控库	主库、系统库；管理当前基线，控制对基线的变更，包含配置单元和集成组件。	复制自由，但变更需权限
静态库	软件仓库、软件产品库；存档已发布的基线。	不可修改

属于**软件管理和软件支持工具**范畴，包含项目管理、配置管理等功能，用于支撑SCM的版本控制、变更控制等核心活动。

软件质量管理

- **关注点与目标**：聚焦于一开始避免缺陷产生，目标包括：
  - 事前预防，侧重缺陷预防而非检查；

    - 刚引入缺陷时即捕获，避免缺陷扩散到下一阶段；
    - 作用于过程而非最终产品，带来广泛影响与巨大收益；
    - 贯穿所有活动，而非集中于某一点。
- **主要任务**：
    - SQA审计与评审：审计软件工作产品、工具和设备，评价是否符合组织标准；
    - SQA报告：记录工作结果并发布，遵循“SQA与高级管理者直接沟通、发布给软件工程组、向关心质量的人发布”的原则；
    - 处理不符合问题：是SQA的重要任务。
- **质量认证**：国内软件企业主要采用ISO 9000和能力成熟度模型（CMM）。

软件风险管理

- **风险定义**：是一种不确定的事件或条件，一旦发生会对项目目标产生正面或负面的影响。
- **风险管理活动过程**：

步骤	说明
风险管理计划编制	规划风险管理的整体流程、方法和资源。
风险识别	识别项目中可能存在的风险。
风险定性分析	对识别出的风险进行定性评估，判断其影响程度和发生概率。
风险定量分析	对风险进行定量分析，量化其影响和概率。
风险应对计划编制	制定应对风险的策略和措施。
风险监控	持续监控风险的状态，确保应对措施有效。

逆向工程

概念	定义
逆向工程	分析已有程序，寻求比源代码更高层的抽象表现形式。
重构	同一抽象级别上转换系统描述形式。
设计恢复	借助工具从已有程序中抽象出数据设计、总体结构设计和过程设计等方面的信息。
再工程	对现有系统的重新开发过程，包含**逆向工程**、**新需求考虑**、**正向工程**三个步骤。
正向工程	不仅从现有系统恢复设计信息，还利用该信息改变或重构系统，以提升整体质量。

逆向工程完备性分级

级别	核心内容
实现级	包含程序的抽象语法树、符号表、过程的设计表示。
结构级	包含反映程序分量间相互依赖关系的信息（如调用图、结构图、程序和数据结构）。
功能级	包含反映程序段功能及程序段间关系的信息（如数据和控制流模型）。
领域级	包含程序分量/实体与应用领域概念间对应关系的信息（如实体关系模型）。

现有系统可通过**逆向工程**提取设计信息，结合**新需求**，经**正向工程**构建新系统；也可通过**再工程**直接完成从现有系统到新系统的重新开发。

软件设计原则

设计原则	重要特性
单一职责原则	类的职责要单一，不能将太多职责放在一个类中，设计单一职责的类（如“三个和尚”的故事体现职责分散的问题）。
开闭原则	对扩展开放，对修改关闭，即在不修改原有软件实体的基础上扩展其功能。
里氏替换原则	父类出现的地方，子类可完全替换父类；子类不能覆盖父类非抽象方法，可增加特有方法；子类重载父类方法时前置条件更宽松，实现父类抽象方法时后置条件更严格（如“龙生龙凤生凤”的类比）。
依赖倒转原则	针对抽象进行编程，而不要针对具体的类。
接口隔离原则	使用多个专门的接口取代统一接口，需关注接口的个数和粒度问题（如门的开关、报警等专门接口的设计）。
合成复用原则	系统中尽量多用组合/聚合，少用继承（继承耦合性强）；组合关系表现为A类拥有B类实例作为属性或接口。
迪米特法则	一个对象应当对其他对象尽可能少的了解（“不要和陌生人说话”），可借助第三者转发交互。

软件维护

维护类型	核心说明	形象类比
改正性维护	识别和纠正软件错误、性能缺陷，排除实施中的误使用，进行诊断和改正错误的过程。	知错能改
适应性维护	外部环境（硬、软件配置）或数据环境（数据库、数据格式等）变化时的维护。	与时俱进
完善性维护	满足用户新的功能与性能要求，修改或再开发软件以扩充功能、增强性能等。	锦上添花
预防性维护	预先提高软件的可维护性、可靠性，采用先进软件工程方法对软件或部分进行重新设计、编码和测试，为未来改进打基础。	防患于未然

系统转换

转换计划	核心说明	适用场景与特点
直接转换	新系统直接取代现有系统。	风险大，适用于新系统不复杂或现有系统已不能使用；优点是节省成本。
并行转换	新老系统并行工作一段时间，新系统试运行后取代老系统。	风险极小，适用于大型系统；可比较新老系统性能；缺点是耗费人力和时间资源，难以控制数据转换。
分段转换	分期分批逐步转换，是直接和并行转换的集合，将大型系统分为多个子系统依次试运行。	适用于大型项目；耗时，且现有系统和新系统混合使用时需协调好接口等问题。

遗留系统改造

基本上不能进行修改和演化以满足新的变化了的业务需求的信息系统。具有如下特点和改造策略

- **特点**：
    1. 完成企业许多重要业务管理工作，但不能全部满足要求。
    2. 性能落后，采用的技术过时。

3. 通常是大型软件系统，已融入企业业务运作和决策管理机制，维护困难。
4. 无现代信息系统建设方法管理开发，基本无文档，难以理解。

策略类型	核心说明
淘汰策略	技术含量低、业务价值低；完全淘汰是资源浪费，可通过理解和借鉴其功能帮助新系统设计，降低新系统开发风险。
继承策略	技术含量低，满足企业功能或性能要求但商业价值高，业务紧密依赖；开发新系统时需完全兼容其功能模型和数据模型，新老系统并行一段时间后切换。
改造策略	业务价值高，基本满足企业业务运作和决策支持需要；包括系统功能增强（原有系统基础上增加新要求，不改变系统本身）和数据模型改造（旧数据模型向新数据模型转化）。
集成策略	技术含量高但业务价值低，可能仅完成某个部门业务管理；存在多个此类系统时形成信息孤岛，策略是集成。

关键路径与时间参数

项目中**时间最长的活动顺序**，即从开始到结束的所有路径中，活动历时之和最大的路径；进度网络图中可能有多条关键路径。

时间参数解释及公式

概念	解释及相关公式
最早开始时间（ES）	某项活动能够开始的最早时间。
最早结束时间（EF）	某项活动能够完成的最早时间，公式：`EF = ES + 工期估算`
最迟结束时间（LF）	为使项目按时完成，某项工作必须完成的最迟时间
最迟开始时间（LS）	为使项目按时完成，某项工作必须开始的最迟时间，公式：`LS = LF - 工期估算`
总时差（TF）	本活动的最迟开始时间 - 最早开始时间；是不影响总工期的机动时间，公式：`TF = LS - ES = LF - EF`
自由时差（FF）	不影响后续工作的机动时间，公式：`FF = min(紧后工作的ES - 此活动的EF)`

关键路径上的活动（关键活动），其**最早开始时间=最迟开始时间**，**最早结束时间=最迟结束时间**，且**总时差、自由时差均为0**。

紧前关系绘图法（PDM）

又称**前导图法、单代号网络图、活动节点图（AON）**，以**方框/长方形节点**代表活动，节点间用箭头连接以显示逻辑关系。

正推法（计算ES、EF）：从项目开始往项目结束方向计算，取紧前活动的**最大EF**作为当前活动的ES，再通过`EF = ES + 工期估算`计算EF。
逆推法（计算LS、LF）：从项目结束往项目开始方向计算，取紧后活动的**最小ES**作为当前活动的EF，再通过`ES = EF - 工期估算`计算ES。

三点估算：（最可能 * 4 + 最乐观 + 最悲观）/ 6 = 节点所需天数

数据库基础知识

数据库基本概念

概念	说明
数据（Data）	数据库中存储的基本对象，是描述事物的符号记录，类型包括文本、图形、图像、音频、视频等。
数据库（DB）	统一管理、长期存储在计算机内的有组织的相关数据集合。 特征： - 按一定数据模型组织、描述和储存； - 数据间联系密切、冗余度较小； - 数据独立性较高； - 易扩展； - 可为各种用户共享。
数据库管理系统（DBMS）	数据库系统的核心软件，由一组相互关联的数据集合和访问数据的软件组成，是解决数据组织存储、高效获取维护的系统软件。 主要功能：数据定义（DDL）、数据操纵（DML）、数据库运行管理、数据库建立与维护。

数据模型

- **三要素**：
    - 数据结构：对象类型的集合，描述系统静态特性。
    - 数据操作：对数据库对象实例的操作集合（检索、插入、删除、修改等）及操作规则。
    - 数据约束条件：一组完整性规则的集合，约束应用数据的语义。
- **常见基本数据模型**：
    - 层次模型：树形结构表示数据间联系。
    - 网状模型：网络结构表示数据间联系。
    - 关系模型：二维表结构。
    - 面向对象数据模型：第三代数据库系统，融合面向对象技术。

数据库管理系统（DBMS）功能

- 数据定义：提供数据定义语言（DDL），描述数据库结构（外模式、模式、内模式）、完整性、安全保密等，定义存储在数据字典中。
- 数据库操作：提供数据操纵语言（DML），实现数据的检索、插入、修改和删除。
- 数据库运行管理：负责并发控制、安全性检查、完整性执行、日志管理、事务管理和自动恢复等。
- 数据组织、存储和管理：管理数据的物理存储和逻辑组织。
- 数据库的建立和维护：包括数据库初始建立、数据转换、转储恢复、重组重构、性能监测分析等。

数据库管理系统（DBMS）特点

- 数据结构化且统一管理。
- 数据独立性高。
- 数据控制功能：
    - 安全性：保护数据库免受非法使用造成的泄露、更改或破坏。

    - 完整性：保证数据库数据的正确性和相容性，防止非法更新。
    - 并发控制：协调多用户并发事务的执行，保证数据库完整性。
    - 故障恢复：在故障导致数据库状态不一致时，将其恢复到正确状态。

三级模式两级映像

模式类型	说明
概念模式	数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图；一个数据库只有一个概念模式。
外模式	子模式、用户模式，描述用户看到或使用的部分数据的逻辑结构；用户通过外模式操作数据库数据；一个数据库可有多个外模式。
内模式	定义存储记录的类型、存储域表示及物理顺序，包括索引、存储路径等数据存储组织；一个数据库只有一个内模式。

- **两级映像**：
    - 外模式/概念模式映像：保证数据的**逻辑独立性**（概念模式改变时，外模式可通过映像保持不变，应用程序无需修改）。
    - 概念模式/内模式映像：保证数据的**物理独立性**（内模式改变时，概念模式可通过映像保持不变，应用程序无需修改）。

关系表类型

关系表类型	要点
基本关系（基表）	实际存在的表，是实际存储数据的逻辑表示。
查询表	查询结果对应的表。
视图表	由基表或其他视图表导出的虚表，本身不独立存储，仅存放定义；优点：简化用户操作、多视角看数据、提供逻辑独立性、保护机密数据。 物化视图：实体化的视图，本身存储数据，原始数据更新时同步更新。

关系数据库基本概念

术语	说明
属性（Attribute）	描述事物的特征，每个属性对应一个取值范围（域）。
域（Domain）	属性的取值范围，所有域应为原子数据。
目（Degree）	关系中属性的个数。
候选码（Candidate Key）	能唯一标识一个元组的属性或属性组。
主码（Primary Key）	从候选码中选定的一个，作为关系的主键。
主属性（Prime Attribute）	包含在任何候选码中的属性；不包含在任何候选码中的属性为非主属性。
外码（Foreign Key）	关系中引用另一关系主码的属性，用于建立关系间的联系。
全码（All-key）	关系的所有属性组构成候选码。

关系模式

关系的描述称为关系模式，形式化表示为五元组 $R(U, D, \mathrm{DOM}, F)$，简记为 $R(A_1, A_2, \ldots, A_n)$，其中：

- R：关系名；
- U：属性名集合；
- D：属性来自的域；
- DOM：属性向域的映象集合；
- F：属性间的数据依赖关系集合。

完整性约束

约束类型	说明
实体完整性	实体的主属性（包含在候选码中的属性）不能取空值，且具有唯一性。
参照完整性	若关系A的属性是关系B的主码，则关系A中该属性要么为空，要么必须在关系B的主码中存在。
用户定义完整性	反映具体应用的数据约束条件，如年龄范围（1-50）、性别取值（男/女）等。

相关数据库对象

- **触发器**：由增删改操作触发的特殊过程，可进行复杂数据检查和操作；`select` 操作无法激活触发器，且触发器中不包含事务控制语句。
- **存储过程**：预定义并编译的一段SQL语句（可含流程控制），存储在数据库服务器上供应用程序调用。

关系代数

集合运算符

运算符	含义	名词解释
∪	并	关系R与S的并由属于R或属于S的元组构成。
−	差	关系R与S的差由属于R但不属于S的元组构成。
∩	交	关系R与S的交由属于R且属于S的元组构成。
×	笛卡尔积	n目关系R和m目关系S的笛卡尔积是(n+m)列的元组集合，前n列是R的元组，后m列是S的元组。

专门的关系运算符

运算符	含义	名词解释
σ	选择	选取关系R中满足条件的行。
π	投影	选取关系R中满足条件的列。
⋈	连接	等值连接（=）：R和S的笛卡尔积中属性值相等的元组。 自然连接（⋈）：特殊的等值连接，要求比较的属性组相同，且结果去掉重复属性。

连接的特殊类型

类型	说明
左外连接	R和S自然连接时，将R中舍弃的元组加入新关系。
右外连接	R和S自然连接时，将S中舍弃的元组加入新关系。
完全外连接	R和S自然连接时，将R和S中舍弃的元组都加入新关系。

除运算（÷）例题解析：设关系R、S如图(a)、(b)，求R÷S的结果。

R（A,B,C）			S（B,C,D）		
a_1	b_1	c_2	b_1	c_2	d_1
a_2	b_3	c_7	b_2	c_1	d_1
a_3	b_4	c_6	b_2	c_3	d_2
a_1	b_2	c_3			
a_4	b_6	c_6			
a_2	b_2	c_3			
a_1	b_2	c_1			

步骤：

1. 确定R和S的公共属性（B、C）。
2. 找出R中属性A的**象集**（A取值对应的B、C组合）：
    - a_1的象集：{(b_1,c_2), (b_2,c_3), (b_2,c_1)}
    - a_2的象集：{(b_3,c_7), (b_2,c_3)}
    - a_3的象集：{(b_4,c_6)}
    - a_4的象集：{(b_6,c_6)}
3. 找出S中公共属性（B、C）的组合：{(b_1,c_2), (b_2,c_1), (b_2,c_3)}。
4. 只有a_1的象集包含S的B、C组合，因此**R÷S = {a_1}**。

数据库规范化理论

函数依赖分类

类型	定义	示例
非平凡函数依赖	$X \to Y$，且 $Y \not\subseteq X$	(学号,课程号)→成绩
平凡函数依赖	$X \to Y$，且 $Y \subseteq X$	(学号,课程号)→学号
完全函数依赖	$X \to Y$，且X的任意真子集都不能推出Y	(学号,课程号)→成绩（学号或课程号单独无法推出成绩）
部分函数依赖	$X \to Y$，但X的某个真子集能推出Y	(学号,课程号)→系部（学号可单独推出系部）
传递依赖	$X \to Y$，$Y \to Z$，则X传递依赖于Z	学号→系部，系部→系主任，则学号传递依赖于系主任

Armstrong公理及推论

规则类型	名称	内容
基本规则	自反律	若 $Y \subseteq X \subseteq U$，则 $X \to Y$ 为函数依赖集F所蕴含
基本规则	增广律	若 $X \to Y$ 为F所蕴含，且 $Z \subseteq U$，则 $XZ \to YZ$ 为F所蕴含
基本规则	传递律	若 $X \to Y$、$Y \to Z$ 为F所蕴含，则 $X \to Z$ 为F所蕴含
推论	合并规则	若 $X \to Y$、$X \to Z$，则 $X \to YZ$ 为F所蕴含
推论	伪传递规则	若 $X \to Y$、$WY \to Z$，则 $XW \to Z$ 为F所蕴含
推论	分解规则	若 $X \to Y$，$Z \subseteq Y$，则 $X \to Z$ 为F所蕴含

不规范化的四大问题

问题类型	说明	示例
数据冗余	数据被重复存储	某课程有100个学生选修，教师信息重复存储100次
修改异常	修改导致数据不一致	修改教师地址时，需同步修改所有关联元组，否则地址不一致
插入异常	缺少部分信息时无法插入或需插入空值	无听课学生名单时，教师授课信息无法插入数据库
删除异常	删除了不该删除的关联数据	删除学生选课信息时，连带删除课程、教师等信息

各级范式定义与升级

范式	定义	升级条件/示例
1NF	关系模式的每个属性分量不可再分（原子性）	如关系`S(name, address(street, city))`不满足1NF，需分解为`S(name, street, city)`
2NF	属于1NF，且非主属性完全依赖于主键	如`R(学号,姓名,班级,课程,成绩)`，主键(学号,课程)，姓名、班级部分依赖于学号，需分解为`R1(学号,姓名,班级)`、`R2(学号,课程,成绩)`
3NF	属于2NF，且消除非主属性对主键的传递函数依赖	如`Store(商店,商品,经营部,经理)`，(商店,商品)→经营部，(商店,经营部)→经理，存在传递依赖，需分解为`R1(商店,商品,经营部)`、`R2(商店,经营部,经理)`
BCNF	每个函数依赖的决定因素都包含候选码	如`STJ(S,T,J)`，`T→J`，`(S,J)→T`，`(S,T)→J`，需分解为`SJ(S,J)`、`TJ(T,J)`以满足BCNF

范式升级流程

1. **1NF**：确保属性原子性，消除可分属性。
2. **2NF**：在1NF基础上，消除非主属性对主键的**部分函数依赖**。
3. **3NF**：在2NF基础上，消除非主属性对主键的**传递函数依赖**。
4. **BCNF**：在3NF基础上，消除主属性对主键的**部分/传递函数依赖**，确保所有函数依赖的决定因素包含候选码。

数据库设计阶段

阶段划分与核心任务

阶段	概念描述	可交付成果/关键输出
需求分析	调研用户数据与处理需求，整理为需求规格说明书。	数据流图、数据字典（含数据项、数据流、数据存储、数据加工）、需求说明书
概念结构设计	基于需求分析，抽象为独立于DBMS的概念模型（E-R模型）。	E-R模型；过程包含抽象数据、设计局部E-R图、合并冲突、消除冗余生成基本E-R图
逻辑结构设计	将概念模型转换为特定DBMS支持的逻辑模型（如关系模型）。	关系模式；过程包含ER转关系模式、关系规范化、模式优化、设计用户子模式
物理设计	结合DBMS、硬件和OS特性，设计存储结构、数据存储安排、访问方法等。	存储记录结构、数据存储安排、访问方法设计、完整性与安全性分析、数据库程序设计

概念结构设计（E-R模型设计）

- **集成方法**：
  - 多个局部E-R图**一次性集成**
  - 逐步集成（累加式集成两个局部E-R图）
- **冲突类型**：

冲突类型	说明	示例
属性冲突	含属性域冲突（如学校编码规则不同）、属性值冲突（如重量单位千克/磅）	不同部门对“学生编号”编码规则不一致
结构冲突	同一对象在不同应用中抽象不同（如职工在某应用中是实体，在另一应用中是属性）；同一实体属性个数/顺序不同	职工在人事系统中是实体（含工号、姓名等），在考勤系统中是属性（仅工号）
命名冲突	同名异义（同一名称含义不同）、异名同义（不同名称含义相同）	“客户”与“顾客”异名同义；“订单编号”在不同系统中含义不同（同名异义）

逻辑结构设计（ER模型转关系模式）将E-R模型中的**实体和联系**转换为关系模式，不同联系的转换规则如下：

联系类型	转换规则	示例
1:1联系	方法1：转换为独立关系模式，主键为任一端主键； 方法2：与任意一端关系模式合并，主键保持不变	教师与班级的“管理”联系： 方法1：`管理(职工号，班级号，...)` 方法2：`班级(班级号，学生人数，职工号，...)` 或 `教师(职工号，姓名，班级号，...)`
1:n联系	方法1：转换为独立关系模式，主键为多端主键； 方法2：与n端关系模式合并，主键保持不变	班级与学生的“组成”联系： 方法1：`组成(学号，班级号，...)` 方法2：`学生(学号，姓名，班级号，...)`
n:m联系	转换为独立关系模式，主键为参与双方的主键组合（联合主键）	学生与课程的“选修”联系：`选修(学号，课程号，成绩...)`，主键为`(学号，课程号)`
多元联系	转换为独立关系模式，主键为参与各方的主键组合（联合主键）	课程、教师、教科书的“讲授”联系：`讲授(职工号，课程号，书号...)`，主键为`(职工号，课程号，书号)`

用户视图与反规范化

内容	说明
用户视图确定	根据数据流图确定处理过程视图，根据用户类别确定不同用户视图，提升数据安全性与独立性。
反规范化	为加速读操作性能，选择性添加冗余数据。 操作：冗余列、派生列、表重组、表分割（水平/垂直分割）。 解决冗余不一致的方法：应用程序同步、批量处理同步、触发器同步。
物理结构设计	为逻辑数据模型选取适配的物理结构（存储结构+存取方法），**步骤包括确定数据分布、存储结构、访问方式**。

数据库实施与运行维护

阶段	内容
数据库实施	步骤：建立数据库结构、数据加载（组织数据入库）、试运行和评价。
数据库运行维护	内容：数据库性能监测与改善、数据库备份及故障恢复、数据库重组和重构。

数据库备份

分类维度	类型	说明
数据库状态	冷备份	数据库正常关闭时进行，备份过程不允许存取、修改操作。
	热备份	备份期间允许对数据库进行存取或修改，备份与用户事务可并发执行。
备份内容	完全备份	备份全部文件，不依赖文件存档属性。
	差分备份	备份自上一次完全备份以来变化的文件。
	增量备份	备份上一次备份后（无论何种备份）所有变化的文件。

应用程序与数据库的交互方式

方式类型	说明	示例/工具
库函数	通过数据库提供的函数库访问数据。	Oracle OCI（Oracle调用接口）
嵌入式SQL	将SQL语句直接嵌入高级程序语言中。	-
通用数据接口标准	跨数据库的通用访问接口。	ODBC、JDBC、DAO、RDO、ADO等
对象关系映射（ORM）	解决面向对象与关系数据库的不匹配，将对象与数据库表关联。	Hibernate（全自动）、MyBatis（半自动）、JPA（Java自带）

NoSQL数据库概述（按存储方式分类）

分类	典型产品	应用场景	优点	缺点
文档存储	MongoDB、CouchDB	Web应用，存储文档和半结构化数据	结构灵活，可根据value建索引	缺乏统一查询语法；无事务处理能力
键值存储	Memcached、Redis	内容缓存（会话、配置等）	扩展性好，性能高	数据无结构化，通过键查询值
列存储	Bigtable、HBase、Cassandra	分布式数据存储和管理	可扩展性强，查找速度快，复杂性低	功能局限；不支持事务强一致性
图存储	Neo4j、OrientDB	社交网络、推荐系统、系统图谱	支持复杂图形算法	复杂性高，支持数据规模有限

NoSQL框架

层级	核心内容
数据持久层	定义数据存储形式，包含4种类型： - 基于内存：存取速度最快，但可能数据丢失； - 基于硬盘：存储持久但存取速度慢； - 内存+硬盘结合：兼顾速度与数据不丢失； - 订制可插拔：数据存取灵活性高。
数据分布层	定义数据分布机制，主要有3种形式： - CAP支持（**C：一致性**、**A：可用性**、**P：分区容错性**）：支持水平扩展； - 多数据中心支持：跨多数据中心平稳运行； - 动态部署支持：集群中可动态添加/移除节点。
数据逻辑模型层	表述数据的逻辑表现形式。
接口层	为上层应用提供数据调用接口，有5种选择：Rest、Thrift、Map/Reduce、Get/Put、特定语言API。

原理	全称	解释	在分布式系统中的角色
C	**一致性**	所有节点看到的数据完全相同	**与 A 二选一（当 P 发生时）**
A	**可用性**	每个请求都能得到响应	**与 C 二选一（当 P 发生时）**
P	**分区容错性**	系统能容忍网络中断	**必须接受**

核心思想：在分布式系统中，当不可避免的网络分区故障发生时，你必须在**强一致性（C）** 和**高可用性（A）** 之间做出痛苦的权衡。这就是 CAP 原理的精髓。

数据库事务相关

事务的ACID属性

特性	说明
原子性（Atomicity）	事务是不可分割的工作单位，操作序列要么全执行，要么全不执行。
一致性（Consistency）	事务执行使数据库从一个一致性状态转变为另一个一致性状态。
隔离性（Isolation）	事务执行过程中不会被其他事务干扰。
持续性（永久性，Durability）	事务一旦提交，其对数据库的改变是永久性的。

以下是SQL标准定义的四个隔离级别，以及它们允许或禁止的并发问题。

隔离级别	脏读	不可重复读	幻读	典型数据库默认级别
读未提交 (Read Uncommitted)	**可能**	**可能**	**可能**	（很少用作默认）
读已提交 (Read Committed)	**不可能**	**可能**	**可能**	**Oracle, PostgreSQL**
可重复读 (Repeatable Read)	**不可能**	**不可能**	**可能**	**MySQL (InnoDB)**
可串行化 (Serializable)	**不可能**	**不可能**	**不可能**	（某些场景下使用）

- **脏读**：一个事务读到了另一个**未提交事务**修改的数据。**读已提交** 及更高级别的隔离级别都可以防止脏读。
- **不可重复读**：在同一个事务中，两次读取**同一个数据项**，得到的结果不同。这是因为在两次读取之间，另一个事务**修改并提交**了该数据。**可重复读** 及更高级别的隔离级别可以防止不可重复读。通常通过在事务期间对**已读取的行**加锁来实现。
- **幻读**：在同一个事务中，两次执行**相同的查询**，返回的**结果集**行数不同。这是因为在两次查询之间，另一个事务**插入或删除**了符合查询条件的行并提交。**与不可重复读的区别**：不可重复读针对的是**已存在的某一行**的值变化；幻读针对的是**结果集的行数**变化（新增或删除了行）。 只有 **可串行化** 隔离级别能真正防止幻读。它通过范围锁或表级锁等机制，阻止其他事务插入符合当前事务查询条件的新数据。

加锁技术类型

类型	定义与特点
排他型封锁（X封锁）	事务T对数据A加X锁后，仅允许T读取和修改A，其他事务需等T解除X锁后才能对A加任何锁；具有排他性，仅允许一个事务独锁数据。
共享型封锁（S封锁）	事务T对数据A加S锁后，允许T读取A但不能修改A；在所有S锁解除前，不允许任何事务对A加X锁；支持并发读，限制修改。

封锁协议（加锁）

级别	内容	优点	缺点
一级封锁协议	事务在修改数据前必须对该数据加X锁，直到事务结束才释放；只读数据可不加锁。	防止“丢失修改”	不加锁的只读事务可能读脏数据或出现“不可重复读”
二级封锁协议	事务修改数据前加X锁（直到事务结束释放）；其他事务读数据前加S锁，读完后即可释放S锁。	防止“丢失修改”、防止“读脏数据”	加S锁的事务可能出现“不可重复读”
三级封锁协议	事务修改数据前加X锁（直到事务结束释放）；其他事务读数据前加S锁，直到事务结束才释放S锁。	防止“丢失修改”、防止“读脏数据”、防止“不可重复读”	-

两段锁协议

- **核心规则**：事务需分两个阶段对数据进行加锁和解锁，可保证调度的**可串行化**，但无法避免死锁。
  - **扩展阶段（获得封锁阶段）**：事务可申请任何数据项的**任意类型锁**，但**不能释放任何锁**。
  - **收缩阶段（释放封锁阶段）**：事务可释放任何数据项的**任意类型锁**，但**不能再申请任何锁**。
- **示例对比**：
  - 遵守两段锁协议的事务T：`Slock A → Slock B → Xlock C → Unlock B → Unlock A → Unlock C`（先集中加锁，后集中解锁）。
  - 不遵守两段锁协议的事务Tj：`Slock A → Unlock A → Slock B → Xlock C → Unlock C → Unlock B`（加锁阶段中途释放锁，违反两段锁规则）。

事务故障与恢复

故障类型	故障原因	解决办法
事务本身的可预期故障	逻辑错误	在程序中预先设置`Rollback`语句，主动回滚事务
事务本身的不可预期故障	算术溢出、存储保护违反	由DBMS恢复子系统通过日志**撤销（UNDO）**事务修改，回退到事务初始状态
系统故障	系统停止运转	通过**检查点法**，系统重启时自动完成恢复
介质故障	外存（如硬盘）破坏	借助日志**重做（REDO）**已提交事务的操作，恢复数据一致性

- **撤销事务（UNDO）**：故障发生时**未完成的事务**，需撤销其对数据库的所有修改。
- **重做事务（REDO）**：故障发生前**已提交的事务**，需重新执行其对数据库的修改，保证数据持久性。

分布式数据库

- **定义**：物理上分散、逻辑上相关的数据库系统，数据分布在计算机网络的不同场地，每个场地具备自治处理能力（可完成局部应用），同时参与全局应用，通过网络通信子系统执行全局操作。
- **特性**：物理分布性、逻辑相关性、场地自治性、场地透明性。

包含六个层次，用于概括分布式数据库的概念和结构：

层次名称	定义与作用	核心特点与映射关系
全局外模式	是全局应用的**用户视图**，为全局用户提供访问分布式数据库的接口，屏蔽全局概念模式的细节。	与集中式数据库的“外模式”功能类似，每个全局用户可对应一个或多个全局外模式；通过“全局外模式→全局概念模式”的映射，实现用户逻辑与全局数据的关联。
全局概念模式	定义分布式数据库中**所有数据的逻辑结构**，是全局数据的“整体蓝图”，描述数据的全局逻辑关系（如实体、属性、联系）。	类似于集中式数据库的“概念模式”，但需体现分布式环境下的数据逻辑相关性；通过“全局概念模式→分片模式”的映射，将全局数据划分为若干分片。
分片模式	是分布式数据库的**特有层次**，负责将“全局概念模式”中的数据**划分为不相交的片段（分片）**，支持水平分片、垂直分片、导出分片或混合分片。	分片需满足“完备性（所有数据都被分片覆盖）、可重构（分片可合并还原全局数据）、不相交（分片间无重叠）”原则；通过“分片模式→分布模式”的映射，确定每个分片的物理分布策略。
分布模式	定义每个**分片在物理场地的分布情况**，即指定每个分片被存储到哪些场地。	决定数据的物理分配策略（如集中式、分割式、全复制式、混合式）；通过“分布模式→局部概念模式”的映射，将分片与具体场地的局部数据模型关联。
局部概念模式	每个场地的**局部数据库的概念模式**，是“全局概念模式”中对应本地数据的**子集的逻辑描述**，用于屏蔽本地存储的细节。	类似于集中式数据库的“概念模式”，但仅描述本地数据的逻辑结构；通过“局部概念模式→局部内模式”的映射，实现本地逻辑数据与物理存储的关联。
局部内模式	每个场地的**局部数据库的物理存储描述**，定义本地数据的物理存储结构（如索引、存储路径、数据文件组织）。	与集中式数据库的“内模式”功能一致，负责本地数据的物理存储管理；是分布式数据库的“物理存储终点”，直接与硬件存储交互。

分布式数据库的六个层次通过**多层映射**实现“逻辑统一、物理分布”：

- 全局用户通过**全局外模式**发起请求，经“全局外模式→全局概念模式”映射，进入全局数据的逻辑层；
- 全局概念模式经**分片模式**划分为分片，再经**分布模式**确定分片的物理场地；
- 每个场地的**局部概念模式**接收对应分片的逻辑描述，最终通过**局部内模式**落地到本地物理存储。

数据分片（4种形式）

分片类型	定义	原则
水平分片	将全局关系的元组划分为不相交的子集。	完备性、可重构、不相交
垂直分片	将全局关系的属性划分为若干子集，每个属性至少映射到一个垂直分片中，且包含主键。	完备性、可重构、不相交
导出分片	水平分片的条件基于其他关系的属性，而非本关系属性。	完备性、可重构、不相交
混合分片	水平、垂直、导出分片的混合形式。	完备性、可重构、不相交

数据分配策略（4种）

策略类型	定义
集中式	所有数据分段都安排在同一个场地。
分割式	数据只有一份，分割成若干逻辑分段，每个分段派到特定场地。
全复制式	数据在每个场地重复存储，每个场地有完整数据副本。
混合式	介于分割式和全复制式之间的分配方式。

分布透明性（三级）

透明性类型	定义
分片透明性	分布透明性的最高层次，用户/应用程序仅操作全局关系，无需考虑数据分片。
位置透明性	用户/应用程序需了解分片情况，但无需了解片段的存储场地。
局部数据模型透明性	最低层次透明性，用户无需关心局部DBMS的数据模型和操纵语言，系统自动完成用户模型与局部模型的转换。

分布式事务协议

两阶段提交协议（2PC）

- **阶段**：
    - 表决阶段：协调者发“准备提交”指令，参与者表决（一票否决权），形成共同决定。
    - 执行阶段：参与者根据协调者指令提交或撤销事务，发送确认信息。
- **规则**：只要有一个参与者撤销事务，协调者必须全局撤销；所有参与者同意，才全局提交。

三阶段提交协议（3PC）

- **阶段**：

    - 第一阶段：协调者发“准备提交”指令，所有参与者表决，全同意才进入下一阶段。
    - 第二阶段：协调者发“全局预提交”指令，所有参与者“准备就绪”才进入下一阶段。
    - 第三阶段：协调者发“全局提交”报文，完成事务提交。
- **优势**：相比2PC减少阻塞风险，增强容错性。

系统架构设计基础知识

软件架构设计概念及生命周期

软件架构设计是通过一系列设计活动，获得满足**系统功能性需求**，并符合**非功能性需求（与质量属性含义相似）**的软件系统框架模型。设计过程中主要考虑系统非功能性需求，软件工程中通过**体系结构风格**、**特定领域架构（DSSA）**等技术实现架构经验的总结与重用。

生命周期阶段（6个阶段）

阶段	核心内容
需求分析阶段	关注“问题空间”到“解空间”的转换，核心是从需求模型构建软件架构模型，并保证模型转换的可追踪性。如何根据需求模型构建软件架构模型；如何保证模型转换的可追踪性；*用例图*
设计阶段	是软件架构研究最关注的阶段，内容包括： - 软件架构模型描述（构件、连接子的组成与组织规则）； - 设计与分析方法； - 架构经验的总结与复用。 细分层次： - SA基本概念（构件+连接子的系统组成）； - 体系结构描述语言（ADL，支持构件、连接子及配置的描述）； - 多视图表示（如4+1视图，从不同视角描述系统架构）。
实现阶段	研究方向： - 基于软件架构的开发过程支持（项目组织、配置管理等）； - 架构到实现的过渡（程序设计语言引入、模型映射、构件组装、中间件复用等）； - 基于架构的测试技术。
构件组装阶段	研究内容： - 支持可复用构件的互联（实现架构设计中连接子的规约）； - 检测并消除体系结构失配问题，包括： ① 构件失配（基础设施、控制/数据模型的假设冲突）； ② 连接子失配（交互协议、数据模型的假设冲突）； ③ 系统成分对全局架构假设的冲突失配。
部署阶段	作用： - 提供高层体系结构视图，描述部署阶段的软硬件模型； - 基于架构模型分析部署方案的质量属性，选择合理部署方案。
后开发阶段	指软件部署后的阶段，研究围绕维护、演化、复用展开，典型方向包括动态软件体系结构、体系结构恢复与重建等。

基于架构的软件设计方法（ABSD）

基于体系结构的软件设计（Architecture-Based Software Design，ABSD）是**体系结构驱动**的方法，由**商业需求、质量需求、功能需求**的组合驱动，用于指导软件系统的架构设计与开发。

- 采用**视角与视图**描述软件架构，**用例**描述功能需求，**质量场景**描述质量需求。
- 是**自顶向下、递归细化**的方法：软件体系结构通过该方法逐步细化，直至产生可实现的软件构件和类。

三个基础

基础类型	说明
功能的分解	使用基于模块的内聚和耦合技术，对系统功能进行拆分。
选择体系结构风格	通过选定合适的体系结构风格（如分层、管道-过滤器等），实现质量和商业需求。
软件模板的使用	利用已有软件系统的结构作为模板，复用成熟的架构设计经验。

软件架构风格

软件体系结构（架构）风格是**特定应用领域中系统组织方式的惯用模式**，定义一个“系统家族”：包含**词汇表**（构件和连接件类型）和**一组约束**（构件与连接件的组合规则）。它反映领域中众多系统共有的**结构和语义特性**，指导模块和子系统的有效组织。

数据流风格

以“数据传递与处理的顺序性”为核心，包含两种子风格：

子风格	定义	优点	缺点	典型实例
批处理序列	每个处理步骤是独立程序，前一步结束后下一步才开始，数据以“整体方式”传递。	- 流程清晰，适合批量数据处理； - 构件独立性强，便于维护。	- 灵活性差，步骤间依赖严格； - 数据必须完整，不支持流式处理。	银行批量对账系统
管道/过滤器	系统分解为序贯处理步骤，步骤间通过“数据流”连接（输出是下一个的输入）；构件是“过滤器”（处理步骤），连接件是“管道”（数据传输）。	- 高内聚、低耦合； - 重用性/可维护性好； - 可扩展性强（标准接口适配）； - 支持并行处理； - 构件隐蔽性好。	- 交互性差，不适合复杂交互场景； - 数据解析/合成开销大，性能受限； - 系统复杂性高。	UNIX Shell脚本、传统编译器

调用/返回风格

以“调用-返回机制”实现分而治之，降低系统复杂度，包含四类子风格：

子风格	定义与核心特性	典型实例
主程序/子程序风格	单线程控制，将问题划分为若干处理步骤；**构件**是主程序和子程序，**连接件**是“过程调用”，调用关系具有层次性。	C语言命令行程序
面向对象风格	基于“数据抽象”和“面向对象”思想，数据与操作封装在“对象”（抽象数据类型实例）中；构件间通过过程调用交互。	Java、Python的面向对象程序
层次风格	系统分层，每层为**上层提供服务**、作为**下层的客户**；内部接口仅对相邻层可见，支持跨层重用。	网络协议栈（TCP/IP分层）、操作系统内核分层
客户端/服务器（C/S）风格	基于“资源不对等”和“共享”设计： - 两层C/S：“胖客户机，瘦服务器”，客户机承担交互与业务逻辑，服务器负责数据管理； - 三层C/S：增加“应用服务器”，业务逻辑驻留于应用服务器，客户机仅负责表示层（“瘦客户机”），分为表示层、功能层、数据层。	传统ERP系统（两层C/S）、Web应用（三层C/S）

优点	缺点	典型实例
1. 良好的重用性，接口不变时组件可在其他场景复用 2. 可维护性好，层次间边界清晰 3. 可扩展性好，支持递增式设计	1. 并非所有系统都适合分层 2. 难以找到合适、正确的层次抽象方法 3. 高耦合系统的层次化实现难度大	1. 网络协议栈（如TCP/IP分层，每层形成功能级虚拟机） 2. 操作系统内核分层（多层协同工作且实现透明性）

仓库（以数据为中心）风格

以“数据”为核心组织系统，包含两类子风格：

子风格类型	定义与核心构件	典型应用场景与实例
数据库系统	构件分为**中央共享数据源**（保存系统数据状态）和**多个独立处理单元**（对数据元素执行操作）。	Oracle数据库开发的管理系统
黑板系统	适用于解决复杂非结构化问题，综合运用多种知识源。由三部分组成： - 知识源：含独立应用相关知识，知识源间仅通过黑板交互； - 黑板数据结构：按应用层次组织问题数据，知识源通过修改黑板数据求解； - 控制：由黑板状态驱动，决定知识的使用。	语音识别、模式识别系统；松耦合代理数据共享存取场景

虚拟机风格

通过构建“虚拟运行环境”增加架构灵活性，包含两类子风格：

子风格类型	定义与核心构件	典型实例
解释器风格	包含解释引擎、被解释代码的存储区、记录解释状态和执行进度的数据结构，用于弥合程序语义与硬件语义的差异。	专家系统
规则系统风格	由规则集、规则解释器、规则/数据选择器及工作内存组成，基于规则对数据进行处理。	基于规则的决策系统

优点	缺点	要点
可以灵活应对自定义场景	复杂度较高	1. 解释器：适用于需要“自定义规则”的场合； 2. 规则为中心：在解释器基础上增加经验规则，适合专家系统。

独立构件风格

独立构件风格强调系统中每个构件相对独立，不直接通信，以降低耦合度、提升灵活性，包含以下两类子风格：

子风格类型	定义与核心特点	优点	缺点
进程通信风格	构件是独立的过程，连接件是**消息传递**；消息传递方式可采用点到点、异步/同步、远程过程调用（RPC）等。	1. 松耦合，构件独立性强； 2. 复用性、可修改性、可扩展性良好。	1. 构件对系统计算的控制能力弱； 2. 数据交换易出现一致性问题； 3. 过程语义依赖事件上下文，正确性推理难度大。
事件系统风格	基于**隐式调用**思想，构件不直接调用过程，而是触发/广播事件；其他构件在事件中注册过程，事件触发时自动调用注册过程。 特点：事件触发者不知道哪些构件会被影响，处理顺序不确定，常结合显式调用补充交互。	同上	同上

闭环风格

- **定义与核心逻辑**：当软件操作物理系统时，软件与硬件形成**反馈循环**，通过输入确定输出，使环境达到新状态。适用于嵌入式系统，涉及连续动作与状态。
- **控制流程对比**：
    - 开环控制系统：给定值→控制器→执行器→被控对象（无反馈，抗干扰能力弱）。

    - 闭环控制系统：给定值与反馈量经比较器→控制器→执行器→被控对象→反馈环节（有反馈，抗干扰能力强）。
- **典型场景**：空调控温恒温机制（通过温度传感器反馈调节制冷/制热）。

C2风格

- **定义**：通过**连接件绑定**的并行构件网络，按一组规则运作。
- **系统组织规则**：
    1. 构件和连接件都有顶部和底部。
    2. 构件顶部连接到连接件底部，构件底部连接到连接件顶部；**构件间不允许直接连接**。
    3. 一个连接件可与任意数量的其他构件、连接件连接。
    4. 两个连接件直接连接时，必须从一个的底部到另一个的顶部。
- **结构特点**：强调构件的并行性与松耦合，通过连接件实现灵活交互。

MDA风格（模型驱动架构）

层次	定义	转换关系
平台独立模型（PIM）	高抽象层次，独立于任何实现技术的模型。	PIM → 一个或多个PSM
平台相关模型（PSM）	为特定实现技术定制，用具体实现构造描述系统。	PSM → 代码
代码	用源代码描述系统，每个PSM转换为代码。	-

- **核心思想**：通过模型转换实现“一次建模，多平台部署”，提升开发效率与可维护性。

架构风格总对比

架构风格名	常考关键字及实例	简介
数据流-批处理	传统编译器，每个阶段产生的结果作为下一个阶段的输入，区别在整体	一个接一个，以整体为单位。
数据流-管道-过滤器	传统编译器，前一个输出是后一个输入，区别在整体	一个接一个，前一个输出是后一个输入。
调用/返回-主程序/子程序	-	显示调用，主程序直接调用子程序。
调用/返回-面向对象	-	对象是构件，通过对象调用封装的方法和属性。
调用/返回-层次结构	-	分层，每层最多影响其上下两层，有调用关系。
独立构件-进程通信	-	进程间独立的消息传递，同步异步。
独立构件-事件驱动（隐式调用）	事件触发推动动作，如程序语言的语法高亮、语法错误提示	不直接调用，通过事件驱动。
虚拟机-解释器	自定义流程，按流程执行，规则随时改变，灵活定义，业务灵活组合	解释自定义的规则，解释引擎、存储区、数据结构。
虚拟机-规则系统	机器人	规则集、规则解释器、选择器和工作内存，用于DSS和人工智能、专家系统。
仓库-数据库	现代编译器的集成开发环境IDE，以数据为中心	中央共享数据源，独立处理单元。
仓库-超文本	IDE，以数据为中心	网状链接，多用于互联网。
仓库-黑板	又称为数据共享风格	语音识别、知识推理等问题复杂、解空间很大、求解过程不确定的软件系统，包含黑板、知识源、控制。
闭环-过程控制	汽车巡航定速，空调温度调节，设定参数，不断调整	发出控制命令并接受反馈，循环往复达到平衡。
C2风格	构件和连接件、顶部和底部	通过连接件绑定在一起按照一组规则运作的并行构件网络。

特定领域软件架构（DSSA）

- **定义**：是在特定领域中为一组应用提供组织结构参考的**标准软件框架**，目标是支持该领域中多个应用的生成。

- **领域类型**：

类型	说明
垂直域	定义特定的系统族，包含多个系统，形成该领域的通用软件架构；属于**相同领域的深入**。
水平域	定义多个系统/系统族中功能区域的共有部分，在子系统级覆盖多系统族的特定功能；属于**不同领域的平移**。

活动	核心目标与内容
领域分析	获得**领域模型**，描述领域中系统之间的共同需求（领域需求）。
领域设计	获得**DSSA**，描述领域模型中需求的解决方案，是适应领域内多个系统需求的高层次设计（非单个系统的设计）。
领域实现	依据领域模型及DSSA，**开发和组织可重用信息**。

角色	职责
领域专家	有经验的用户或软件工程师，负责领域内系统的需求分析、设计、实现及项目管理。
领域分析师	具备知识工程背景的系统分析员，负责领域分析工作。
领域设计人员	有经验的软件设计人员，负责领域设计工作。
领域实现人员	有经验的程序设计人员，负责领域实现工作。

DSSA的建立过程是**并发、递归、反复、螺旋式**的，包含5个阶段：

1. 定义领域范围：明确领域应用需满足的用户需求。
2. 定义领域特定的元素：编译领域字典、领域术语的同义词词典。
3. 定义领域特定的设计和实现需求约束。
4. 定义领域模型和架构。
5. 产生、搜集可重用的产品单元。

包含的两个过程

过程	说明
领域工程	为一组相近/相似的应用建立基本能力与必备基础，覆盖建立可重用软件元素的所有活动。
应用工程	通过重用软件资源，以领域通用体系结构为框架，开发满足用户需求的一系列应用软件。

系统质量属性与架构评估

软件系统质量属性与场景描述

软件系统质量属性（Quality Attribute）是**可测量或可测试的属性**，用于描述系统满足利益相关者（Stakeholders）需求的程度。基于软件生命周期，分为**开发期质量属性**和**运行期质量属性**。

开发期质量属性（软件开发阶段关注）

属性	定义
易理解性	设计被开发人员理解的难易程度。
可扩展性	软件适应新需求或需求变化时增加新功能的能力（也称为灵活性）。
可重用性	重用软件系统或其部分的难易程度。
可测试性	对软件测试以证明其满足需求规范的难易程度。
可维护性	修改缺陷、增加功能、提高质量属性时，识别修改点并实施修改的难易程度。
可移植性	将软件系统从一个运行环境转移到另一个不同运行环境的难易程度。

运行期质量属性（软件运行阶段关注）

属性	定义
性能	软件系统及时提供服务的能力，如速度、吞吐量、容量等。
安全性	兼顾向合法用户提供服务，同时阻止非授权使用的能力。
可伸缩性	用户数和数据量增加时，软件系统维持高服务质量的能力（如通过增加服务器实现）。
互操作性	本软件系统与其他系统交换数据和相互调用服务的难易程度。
可靠性	软件系统在一定时间内持续无故障运行的能力。
可用性	系统在一定时间内正常工作的时间占比，受系统错误、恶意攻击、高负载等影响。
鲁棒性（健壮性/容错性）	软件系统在非正常情况（如非法操作、软硬件故障）下仍能正常运行的能力。

面向架构评估的质量属性

属性	定义与关键要点
性能	系统的响应能力，如响应时间、吞吐量；设计策略包括优先级队列、增加计算资源、引入并发机制等。
可靠性	系统在错误/意外使用下维持功能的能力，用MTTF（平均失效等待时间）、MTBF（平均失效间隔时间）衡量；分为容错（错误发生时内部“修复”）和健壮性（忽略错误输入）；设计策略包括冗余、心跳、选举、Ping/Echo等。
可用性	系统正常运行的时间比例，设计策略同可靠性（冗余、心跳等）。
安全性	向合法用户提供服务的同时阻止非授权使用；分为机密性（信息不泄露给未授权方）、完整性（信息不被非法修改）、不可否认性（双方不能否认信息交换行为）、可控性（控制信息传播）；设计策略包括入侵检测、用户认证、授权、追踪审计等。
可修改性	快速且高性价比地变更系统的能力，分为可维护性（修复错误的难易）、可扩展性（适应新需求的能力）、结构重组（重组构件及关系）、可移植性（环境迁移的难易）；设计策略包括接口-实现分类、抽象、信息隐藏等。
功能性	系统完成预期工作的能力，需多构件协作实现。
可变性	架构经扩充或变更成为新架构的能力，符合预定义规则，适用于系列相关产品的基础架构。
互操作性	系统与其他系统/环境交互的能力，需为外部提供精心设计的软件入口。

质量属性场景是面向特定质量属性的需求，由**6部分组成**：

- 刺激源（Source）：生成刺激的实体（人、计算机系统等）。
- 刺激（Stimulus）：刺激到达系统时的条件。
- 环境（Environment）：刺激发生的条件（如系统过载、正常运行等）。
- 制品（Artifact）：被激励的系统或其部分。
- 响应（Response）：刺激到达后采取的行动。
- 响应度量（Measurement）：对响应的可测量指标，用于测试需求。

系统架构评估详细总结

1. **敏感点与权衡点**

- 敏感点：为实现某一特定质量属性，一个或多个构件具备的特性。
- 权衡点：影响**多个质量属性**的特性（是多个敏感点的集合），例如“加密级别”会同时影响安全性（提高加密级则安全增强）和性能（高加密级会增加处理时间），若对加密消息的时延有严格要求，“加密级别”即为权衡点。

2. **风险承担者（利益相关人）**
   参与系统架构决策、关注自身目标的人员（如软件架构师），需在不同质量需求间进行权衡与调停，确保自身目标实现。
3. **场景**
   从风险承担者视角对“系统交互”的简短描述，用于明确架构评估的质量目标；通过**刺激（事件）**、**环境（事件发生的条件）**、**响应（架构的动作）**三要素描述。

评估方法分类

方法类型	核心说明
基于调查问卷/检查表的方法	类似“需求获取”的问卷调查形式，聚焦架构层面的问题，要求评估人员对领域非常熟悉。
基于场景的评估方法	以“场景”为核心，分析架构对质量需求的满足程度，典型方法包括SAAM和ATAM。
基于度量的评估方法	制定定量指标（如代码行数），建立“质量属性-度量结果”的映射关系，要求评估人员对架构深度熟悉；涉及“建立映射原则→从架构文档提取度量信息→推导系统质量属性”三个活动。

基于场景的典型方法详解

（1）SAAM（软件架构分析方法）

- **定位**：非功能质量属性的架构分析方法，是最早形成文档并广泛应用的架构分析方法。
- **目标**：验证“应用程序属性文档”中的架构假设和设计原则。
- **评估技术**：场景技术——场景描述了系统需支持的活动和状态变化，是架构属性的基础描述载体。
- **质量属性**：所有质量属性可具体化为场景，其中**可修改性**是SAAM分析的核心质量属性。
- **风险承担者**：协调不同参与者的共同关注点，为后续决策奠定基础，达成架构共识。
- **架构描述**：适用于架构的“最终版本”（早于详细设计阶段），以**功能**、**结构**、**分配**为三个核心描述维度。
- **方法步骤**：输入为“问题描述、需求声明、架构描述”；步骤为**场景开发→架构描述→单个场景评估→场景交互评估→总体评估**。
- **应用案例**：空中交通管制系统、嵌入式音频系统、WRCS（修正控制系统）、KWIC（上下文关键词检索系统）等。

（2）ATAM（架构权衡分析方法）

- **定位**：在SAAM基础上发展，聚焦**性能**、**安全性**、**可修改性**、**可用性**的多质量属性权衡。
- **参与者**：评估小组、项目决策者、其他项目相关人（如开发人员、用户代表）。
- **活动领域**：场景和需求收集、体系结构视图和场景实现、属性模型构造和分析、折中决策。

- **评估阶段**：
    - 阶段1：演示——介绍ATAM流程、业务驱动因素、待评估的体系结构。
    - 阶段2：调查和分析——确定关键架构方法、**生成质量属性效用树**、分析体系结构的风险/敏感点/权衡点。
    - 阶段3：测试——验证架构决策的合理性，头脑风暴和有限场景。
    - 阶段4：报告——输出评估结论与建议。
- **核心工具：质量属性效用树**
  用于对“性能、安全性、可修改性、可用性”进行分类和优先级排序，结构为**树根→质量属性→属性分类→质量属性场景（叶子节点）**。例如，“性能”可细分为“响应时间、吞吐量”等分类，再下钻到具体场景（如“系统在1000并发用户下响应时间≤2秒”）。
  效用树采用分层结构，从树根到叶子节点依次为：
    - 树根：“效用”（即质量属性对利益相关者的价值）。
    - 质量属性层：如性能（关注系统的响应速度、吞吐量）、安全性（关注数据保护、访问控制、抗攻击能力等）、可修改性（关注系统变更的难易程度（如新增功能、修改模块的工作量））、可用性（关注系统的故障恢复、持续服务能力）（ATAM 重点关注的四类）。
    - 属性分类层：每个质量属性的子类别（如性能可分为 “数据延迟”“交易吞吐量” 等）。
    - 场景层（叶子节点）：具体的质量属性场景，描述 “谁在什么条件下做什么，期望什么结果”。

成本效益分析法（CBAM）

- **定位**：在ATAM基础上构建，用于对架构设计决策的**成本和收益建模**，通过**投资回报率（ROI）** 选择合适架构。
- **步骤**：
    1. 整理场景：确定场景并划分优先级，选择优先级最高的1/3场景分析。
    2. 细化场景：对每个场景分析最好、最坏情况。
    3. 确定场景优先级：项目干系人投票生成场景权值。
    4. 分配效用：建立“策略-场景-响应级别”的效用表。
    5. 形成对应关系：明确策略、场景、响应级别的关联。
    6. 内插法确定效用：根据效用表推导具体场景的效用。
    7. 计算总收益：统计各架构策略的总收益。
    8. 选择架构策略：通过ROI（收益-成本）排序，选择收益最高的架构。

其他评估方法（仅了解）

方法名称	核心说明
SAEM方法	将软件架构视为“最终产品”和“设计中间产品”，从**外部质量属性（用户定义）** 和**内部质量属性（开发者决定）** 两个角度构建评估模型。流程包括： - 对质量属性进行规约建模； - 为内外部属性创建度量准则（从评估目的、角度、环境出发定义目标并提出度量规则）； - 评估质量属性（数据收集、度量、结果分析）。
SAABNet方法	源于人工智能领域，通过**贝叶斯信念网络（BBN）** 表达和使用定性知识，辅助架构的定性评估；支持不确定、不完整知识的推理。变量分为三类： - 架构质量属性变量（如可维护性、灵活性）； - 度量准则变量（如容错性、响应性）； - 架构特征变量（如继承深度、编程语言）。
SACMM方法	软件架构**修改的度量方法**。
SASAM方法	通过映射和比较**预期架构（设计阶段描述）** 与**实际架构（源代码中执行的架构）**，静态评估软件架构。
ALRRA方法	软件架构**可靠性风险评估方法**，使用动态复杂度准则和动态耦合度准则定义组件、连接件的复杂性，结合失效模式和影响分析（FMEA）技术。
AHP方法（层次分析法）	对**定性问题进行定量分析**的多准则决策方法，通过层次化因素、两两对比、计算权值和总排序，实现复杂决策的量化。
COSMIC+UML方法	基于度量模型评估软件架构**可维护性**，采用统一的COSMIC度量方法，辅助分析架构演化方案的可行性，在开源软件DCMM的UML组件图上验证。

软件可靠性基础知识

软件可靠性

软件可靠性是**软件产品在规定的条件下和规定的时间区间完成规定功能的能力**。

与硬件可靠性的区别

对比维度	软件可靠性	硬件可靠性
复杂性	软件复杂性更高，大部分失效来自软件本身。	复杂性相对较低，失效多源于物理故障。
物理退化	不存在物理退化现象。	失效主要由物理退化导致。
唯一性	软件是唯一的，每个复制版本都一样。	两个硬件不可能完全一样。
版本更新周期	更新周期较快。	更新周期较慢。

定量描述

1. **规定时间类型**：自然时间、运行时间、执行时间（占用CPU时间）。
2. **失效概率**：初始时刻为0，随时间单调递增趋向于1。

3. **可靠度**：软件在规定条件和时间内不发生失效的概率，公式为**可靠度 = 1 - 失效概率**。
4. **失效强度**：单位时间内软件出现失效的概率。
5. **平均失效前时间（MTTF）**：系统从开始运行到第一次故障的平均时间。
6. **平均恢复前时间（MTTR）**：从故障出现到修复成功的平均时间。
7. **平均故障间隔时间（MTBF）**：两次连续故障之间的平均时间，公式为**MTBF = MTTF + MTTR**。
8. **系统可用性**：公式为**系统可用性 = MTTF / (MTTF + MTTR) × 100%**。

串并联系统可靠性计算

- **串联系统**：若各设备可靠性为$R_1, R_2, \ldots, R_n$，则系统可靠性 $R = \prod_{i=1}^{n} R_i$。
- **并联系统**：若各设备可靠性为$R_1, R_2, \ldots, R_n$，则系统可靠性 $R = 1 - \prod_{i=1}^{n} (1 - R_i)$。

可靠性目标：指客户对软件性能满意程度的期望，通常用**可靠度、故障强度、平均失效时间（MTTF）** 等指标描述。

可靠性测试的意义与目的

1. **意义**：
    - 软件失效可能造成灾难性后果。
    - 软件失效在计算机系统失效中占比高。
    - 软件可靠性技术不成熟，加剧了问题的重要性。
    - 是软件费用增长的主要原因之一。
    - 系统对软件依赖性强，软件对生产、社会生活影响大。
2. **目的**：
    - 发现软件在需求、设计、编码等环节的缺陷。
    - 为软件使用和维护提供可靠性数据。
    - 确认软件是否达到可靠性的定量要求。

可靠性测试分类

类型	定义
广义可靠性测试	为评价软件系统可靠性，运用建模、统计、试验、分析和评价等手段实施的测试。
狭义可靠性测试	为获取可靠性数据，按预先确定的测试用例，在软件预期使用环境中实施的**面向缺陷的测试**，以用户使用方式测试软件。

软件可靠性模型

- **定义**：为预计或估算软件的可靠性所建立的**可靠性框图和数学模型**。
- **技术角度的影响因素**：运行剖面（环境）、软件规模、软件内部结构、软件的开发方法和开发环境、软件的可靠性投入。

一个软件可靠性模型通常包含以下部分：

组成部分	说明
模型假设	对实际情况的简化或规范化，例如测试代表实际运行环境、软件失效独立发生等。
性能度量	模型的输出量，如失效强度、残留缺陷数等，通常以数学表达式呈现。
参数估计方法	对于无法直接获得的可靠性度量（如残留缺陷数），通过估计参数间接确定其值。
数据要求	模型需要的输入数据，即软件可靠性数据。

共同假设：

1. 代表性假设：测试产生的可靠性数据可预测运行阶段的软件可靠性行为。
2. 独立性假设：软件失效在不同时刻独立发生，互不影响。
3. 相同性假设：所有软件失效的后果（等级）相同，仅关注失效发生时刻，不区分失效严重等级。

模型分类(了解即可)

模型类型	说明
种子法模型	利用“捕获-再捕获”抽样技术，通过预先“播种”设定错误，根据测试出的原始错误数和诱导错误比例，估计程序中残留的错误数。
失效率类模型	研究程序的失效率。
曲线拟合类模型	用回归分析方法研究软件复杂性、缺陷数、失效率、失效间隔时间等。
可靠性增长模型	预测软件在检错过程中的可靠性改进，用增长函数描述改进过程。
程序结构分析模型	根据程序、子程序及调用关系，形成可靠性分析网络。
输入域分类模型	选取软件输入域的样本“点”运行程序，根据样本点的使用概率和测试成功/失效率，推断软件的使用可靠性。
执行路径分析方法模型	计算程序各逻辑路径的执行概率和错误路径的执行概率，综合得出软件的使用可靠性。
非齐次泊松过程模型	以软件测试中单位时间的失效次数为独立泊松随机变量，预测某使用时间点的累计失效数。
马尔可夫过程模型	-（未详细展开，属于可靠性建模方法之一）
贝叶斯模型	利用失效率的试验前分布和当前测试失效信息，评估软件的可靠性。

软件可靠性管理

是软件工程管理的一部分，以**全面提高和保证软件可靠性**为目标，以**软件可靠性活动**为主要对象，将现代管理理论用于软件生命周期可靠性保障活动的管理形式。

涵盖软件工程各阶段可靠性活动的**目标、计划、进度、任务和修正措施**等。

阶段	主要可靠性活动
需求分析阶段	确定可靠性*目标*、分析影响*因素*、确定验收*标准*、制定管理*框架*、制定文档编写*规范*、制定活动初步*计划*、确定数据收集*规范*。
概要设计阶段	*确定可靠性度量*、*制定详细验收方案*、*可靠性设计*、收集可靠性数据、调整活动计划、明确后续阶段详细计划、编制文档。
详细设计阶段	*可靠性设计*、可靠性预测、调整活动计划、收集可靠性数据、明确后续阶段详细计划、编制文档。
编码阶段	可靠性测试（含单元测试）、排错、调整活动计划、收集可靠性数据、明确后续阶段详细计划、编制文档。
测试阶段	可靠性测试（含集成测试、系统测试）、排错、*可靠性建模*、*可靠性评价*、调整活动计划、收集可靠性数据、明确后续阶段详细计划、编制文档。
实施阶段	可靠性测试（含验收测试）、排错、收集可靠性数据、*调整模型*、*可靠性评价*、编制文档。

软件可靠性设计

可靠性设计是在常规软件设计中，应用各种方法和技术，使程序设计在兼顾用户**功能和性能需求**的同时，全面满足软件的可靠性要求。

设计原则

1. 是**软件设计的一部分**，必须在软件总体设计框架中使用，且不与其他设计原则冲突。
2. 以**提高和保障软件可靠性**为最终目标，需满足提高软件质量的前提。
3. 应**确定软件的可靠性目标**，不能无限扩大化，且排在功能度、用户需求和开发费用之后考虑。

容错设计技术

软件容错技术主要包含**恢复块设计、N版本程序设计、冗余设计**等方法：

- **恢复块设计（动态冗余）**：
  选择一组操作作为容错设计单元，将普通程序块转换为恢复块；是**动态故障屏蔽技术**，采用**后向恢复策略**（系统恢复到前一个正确状态继续执行）。
  设计需保证主块和后备块的独立性，且验证测试程序必须正确。
- **N版本程序设计**：
  设计多个模块或不同版本，对相同输入的操作结果实行**多数表决**，避免某一版本故障导致错误服务；是**静态故障屏蔽技术**，采用**前向恢复策略**（使当前计算继续，恢复成连贯正确状态）。
  要求：软件需求说明需完全精确，且N个版本需由不同人独立设计（使用不同算法、编程语言、工具等），减少表决点的相关错误概率。
- **恢复块与N版本程序设计的对比**：

对比项	恢复块方法	N版本程序设计
硬件运行环境	单机	多机
错误检测方法	验证测试程序	表决
恢复策略	后向恢复	前向恢复
实时性	差	好

- **冗余设计**：
  在完整软件系统外，设计**不同路径、算法或实现方法的模块/系统作为备份**，故障时替换冗余部分以维持系统正常运行。

检错技术

适用于无需在线容错或无法采用冗余设计，但可靠性要求高的场景。

- 特点：实现代价低于容错和冗余技术，但**不能自动解决故障，需人工干预**。
- 设计要素：需着重考虑**检测对象、检测延时、实现方式、处理方式**。

降低复杂度设计

在保证软件功能的前提下，**简化软件结构、缩短代码长度、优化数据流向**，通过降低软件复杂度来提高可靠性。

系统配置技术

通过系统整体架构提供可靠性，主要包含**双机热备技术、服务器集群技术**：

双机热备技术

是软硬件结合的容错方案，由两台服务器、外接共享磁盘阵列柜及双机热备软件组成；采用“心跳”机制（主备系统按时间间隔发送通信信号，监测运行状态）。

模式	说明
双机热备模式（Active/Standby）	一台服务器工作（Active），一台监控准备（Standby），数据实时同步；故障时激活备机，存在计算资源浪费。
双机互备模式	两个独立应用在两台机器同时运行，互为备机；故障时接管对方应用，对服务器性能要求高。
双机双工模式	两台服务器均处于活动状态，运行相同应用；实现负载均衡和互为备份，常用于Web、FTP服务器等。

服务器集群与负载均衡

- **定义**：一组相互独立的服务器在网络中组合成单一系统协同工作，以单一系统模式管理（多台计算机组织起来协同工作）。
- **工作机制**：每台计算机承担部分计算任务和容错任务；某台计算机故障时，集群软件将其隔离，通过负载转嫁机制完成新的负载分担，同时发出警报；通过功能整合和故障过渡实现高可用性和可靠性。

- **特点**：可伸缩性、高可用性、可管理性、高性价比、高透明性。
- **分类**：高性能计算集群、负载均衡集群、高可用性集群。

负载均衡技术：提高集群系统整体处理能力和可靠性，加快响应速度，提高客户端访问成功概率，使所有节点负荷平均，避免局部过载或轻载。

技术类型	说明
基于特定软件的负载均衡（应用层）	利用网络协议的重定向功能（如HTTP重定向），服务器返回重定向响应，客户端重发请求到新地址，实现负载均衡。
基于DNS的负载均衡（传输层）	在DNS服务器中为同一主机名配置多个地址，应答查询时按顺序返回不同解析结果，引导客户端访问不同节点。
基于NAT的负载均衡	将一个外部IP地址映射为多个内部IP地址，动态转换为内部节点地址，引导外部连接请求到对应节点。
反向代理负载均衡	将Internet的连接请求以反向代理方式动态转发给内部网络的多个节点处理。
混合型负载均衡	结合多种负载均衡技术的实现方式。

软件可靠性测试

由可靠性目标确定、运行剖面开发、测试用例设计、测试实施、测试结果分析等主要活动组成。

- **测试步骤**：
    1. 定义软件运行剖面：为软件使用行为建模，开发使用模型，明确测试内容。
    2. 设计可靠性测试用例。
    3. 实施可靠性测试。
- **可靠性数据分类**：

数据类型	说明
失效时间数据	记录发生一次失效所累积经历的时间。
失效间隔时间数据	记录本次失效与上一次失效间的间隔时间。
分组时间内的失效数	记录某个时间区内发生的失效次数。
分组时间的累积失效数	记录到某个区间的累积失效数。

软件可靠性评价

- **评价过程**：包含选择可靠性模型、收集可靠性数据、可靠性评估和预测3个过程。
- **选择可靠性模型的考虑因素**：模型假设的适用性、预测的能力与质量、模型输出值能否满足可靠性评价需求、模型使用的简便性。
- **可靠性数据收集**：
    - 数据类型：主要是**软件失效数据**，在软件测试、实施阶段收集。

    - 解决方法：及早确定所采用的可靠性模型、制订可实施性较强的可靠性数据收集计划、重视软件测试数据的整理和分析、充分利用数据库来完成可靠性数据的存储和统计分析。
- **可靠性评估和预测**：
    - 目的：评估软件系统的可靠性状况，预测将来一段时间的可靠性水平。
    - 需解答的问题：①判断是否达到了可靠性目标；②如未达到，要再投入多少；③软件系统投入实际运行一段时间后，经过维护、升级和修改，能否达到交付或部分交付用户使用的可靠性水平。
    - 方法：以**软件可靠性模型分析**为主，以失效数据的图形分析法、试探性数据分析技术（EDA）等为辅。

软件架构的演化和维护

软件架构演化

- **定义**：对架构进行修改和完善的迭代过程，目的是使软件适应环境变化，进行纠错性和完善性修改，直至满足用户需求。
- **本质**：软件整体结构的演化，涵盖**全生命周期**，包括软件架构需求获取、建模、文档、实现及维护等阶段。
- **重要性**：
    - 架构是系统的“骨架”，保障软件系统的良好特性。
    - 作为软件蓝图，为宏观管控系统的整体复杂性和变化性提供有效途径。
- **降低演化成本的原因**：
    - 形式化、可视化的架构表示提高了软件的可构造性，便于演化。
    - 架构设计方案涵盖的整体结构、配置、约束等信息，有助于开发人员提前考虑未来演化问题。
    - 架构设计中对组件耦合的描述，利于软件系统的动态调整。

软件架构定义包含**组件**、**连接件**、**约束**三大要素，演化主要关注这三者之间的**添加**、**修改和删除**。

面向对象软件架构的演化类型

对象演化

- **AddObject（AO）**：在顺序图中添加新对象，用于实现新功能或独立现有对象的功能以增加架构灵活性。
- **DeleteObject（DO）**：删除顺序图中现有对象，用于移除功能或合并对象以降低架构复杂度。

消息演化

类型	说明
AddMessage（AM）	增添新消息，用于对象间增加新的交互行为。
DeleteMessage（DM）	删除当前消息，是AM的逆向演化，用于移除交互行为。
SwapMessageOrder（SMO）	交换两条消息的时间顺序，用于改变交互行为之间的关系。
OverturnMessage（OM）	反转消息的发送与接收对象，用于修改交互行为本身。
ChangeMessageModule（CMM）	改变消息的发送或接收对象，用于修改交互行为本身。

复合片段演化

复合片段是对象交互关系的控制流描述，演化类型包括：

类型	说明
AddFragment（AF）	在消息上新增复合片段，用于增添新的控制流。
DeleteFragment（DF）	删除现有复合片段，是AF的逆向演化，用于移除控制流。
FragmentTypeChange（FTC）	改变复合片段类型，用于改变控制流（伴随条件、内部执行序列的演化）。
FragmentConditionChange（FCC）	改变复合片段内部执行条件，用于修改控制流的执行条件。

约束演化

- **Add Constraint（AC）**：直接添加新的约束信息，需判断当前设计是否满足新约束。
- **Delete Constraint（DC）**：移除某条约束信息，用于去除不必要的条件。

演化方式的分类

分类维度	具体类型
实现方式和实施粒度	基于过程和函数的演化、面向对象的演化、基于组件的演化、基于架构的演化。
研究方法	支持演化（如代码模块化、重构）、版本和工程管理工具、架构变换形式方法、成本收益分析。
是否处于系统运行时期	静态演化（设计时、运行前）、动态演化（运行时）。

包括**设计时演化、运行前演化、有限制运行时演化、运行时演化**。

静态演化

- **维护方法**：更正性维护、适应性维护、完善性维护。
- **演化步骤**：
    1. 软件理解：查阅文档，分析架构，识别系统组成元素及关系。
    2. 需求变更分析：找出新需求与原有需求的差异。
    3. 演化计划：分析原系统，确定演化范围和成本，选择演化计划。
    4. 系统重构：根据演化计划重构系统。
    5. 系统测试：测试演化后的系统，查找错误和不足。

一次完整的软件架构演化过程由**一系列原子演化操作组合而成**。原子演化操作是基于UML模型表示的软件架构，在**逻辑语义上粒度最小的架构修改操作**，每经过一次原子演化操作，架构会形成一个演化中间版本。

- **可维护性度量**：基于组件图表示的软件架构，在较高层次上评估原子修改操作（如增加/删除模块间依赖、接口、模块，拆分/聚合模块等）对整个架构的影响。
- **可靠性评估**：基于用例图、部署图和顺序图，分析原子演化操作（如增加/删除消息、交互对象、消息片段、用例执行、角色等）对交互场景可靠程度的影响。

动态演化

- **定义**：在系统运行期间的演化，需在不停止系统功能的情况下完成，发生在**有限制的运行时演化和运行时演化阶段**。
- **需求来源**：
  - 软件内部执行导致的体系结构改变（如服务器端软件在客户请求到达时创建新组件）。
  - 软件系统外部请求导致的重配置（如操作系统升级时无需重启完成体系结构修改）。
- **动态性级别**：

级别	说明
交互动态性	要求数据在固定的结构下动态交互。
结构动态性	允许对结构进行修改（如增删组件和连接件实例），是研究和应用的主流。
架构动态性	允许软件架构的基本构造变动（如新的组件类型定义）。

- **动态演化的方面**：

方面	说明
属性改名	运行中重新定义非功能属性（如服务响应时间）。
行为变化	用户需求变化或系统服务质量调节引发软件行为变化。
拓扑结构改变	增删组件、连接件，改变组件与连接件的关联关系等。
风格变化	一般保持架构风格不变，若改变只能变为衍生风格（如两层C/S到三层C/S）。

- **实现技术**：
  - **动态软件架构（DSA）**：运行时刻系统框架结构会变化，允许运行中通过框架结构动态演化修改架构。
  - **动态重配置（DR）**：从组件和连接件的配置入手，允许运行中增删组件、连接件，修改连接关系等。
- **实现原理**：使DSA在运行应用系统中以有状态、有行为、可操作的实体显式表示，被整个运行环境共享；运行时刻体系结构信息的改变可触发、驱动系统自身的动态调整。
- **系统需提供的功能**：保存当前软件架构信息、设置监控机制监视系统需求变化、保证演化操作原子性。
- **DSA实施动态演化的步骤**：①捕捉并分析需求变化；②获取或生成体系结构演化策略；③选择并实施演化策略；④演化后的评估与检测。
- **基于动态重配置的演化**：软件部署后对配置信息的修改，用于系统动态升级；涉及的修改包括简单任务实现修改、工作流实例任务增删等；动态重配置模式有主从模式、中央控制模式、客户端/服务器模式、分布式控制模式。

软件架构演化原则：演化成本控制原则；进度可控原则；风险可控原则；主体维持原则（保证软件系统主体行为稳定）；系统总体结构优化原则；平滑演化原则（演化速率趋于稳定）；目标一致原则；模块独立演化原则；影响可控原则；复杂性可控原则；有利于重构原则；有利于重用原则；设计原则遵从性原则（判断架构设计原则是否被破坏）；适应新技术原则；环境适应性原则；标准依从性原则；质量向好原则；适应新需求原则

软件架构演化评估方法

评估类型	说明
演化过程已知的评估	目的是通过对架构演化过程进行度量，比较架构内部结构差异及由此导致的外部质量属性变化，评估相关质量属性。
演化过程未知的评估	-

执行过程（演化过程已知时）：一次完整演化前后的架构为A_0和A_n，每经过一次原子演化得到中间版本A_i。对每个中间版本A_i进行度量，得到其质量属性度量值Q_i，$D(i-1,i)$表示版本间的质量属性距离。

基于度量的评估方法：通过对演化前后的软件架构进行度量，比较架构内部结构差异及由此导致的外部质量属性变化；具体包括**架构修改影响分析、监控演化过程、分析关键演化过程。演化过程未知时的处理**：无法追踪演化过程的每一步变化，只能根据架构演化前后的度量结果**逆向推测**架构发生的改变，并分析这些改变与架构相关质量属性的关联关系。

软件架构维护

软件架构维护过程一般涉及**架构知识管理、架构修改管理和架构版本管理**三个方面。

架构知识管理

- **定义**：对架构设计中隐含的决策来源进行文档化表示，帮助维护人员在架构维护过程中完善考虑修改，并为其他软件架构相关活动提供参考。
- **架构知识的构成**：架构知识 = 架构设计 + 架构设计决策（需说明架构设计时采用该架构的原因）。
- **管理侧重**：侧重于软件开发和实现过程所涉及的架构静态演化，从架构文档等信息来源中捕捉架构知识，进而记录和评价架构的质量属性及其设计依据。

架构修改管理

- **核心做法**：建立隔离区域，保障该区域内的修改对其他部分的影响较小甚至无影响。
- **实施要求**：需明确修改规则、修改类型，以及可能的影响范围和副作用等。

架构版本管理

- **作用**：为软件架构演化的版本演化控制、使用和评价等提供可靠依据，并为架构演化量化度量奠定基础。

未来信息综合技术

信息物理系统（CPS）

信息物理系统（Cyber-Physical Systems，CPS）是**控制系统、嵌入式系统的扩展与延伸**，底层技术源于嵌入式技术的应用与提升。

- 技术集成：通过集成感知、计算、通信、控制等信息技术和自动控制技术，构建**物理空间与信息空间中人、机、物、环境、信息等要素相互映射、适时交互、高效协同的复杂系统**，实现系统内资源配置和运行的按需响应、快速迭代、动态优化。
- 本质：构建信息空间与物理空间之间**基于数据自动流动的状态感知、实时分析、科学决策、精准执行的闭环赋能体系**，解决生产制造、应用服务中的复杂性和不确定性问题，提高资源配置效率，实现资源优化。

层级	说明
单元级CPS	CPS的最小不可分割单元，具备可感知、可计算、可交互、可延展、自决策功能；例如智能部件、工业机器人、智能机床。
系统级CPS	多个单元级CPS通过工业网络（如工业现场总线、工业以太网）互联，实现更大范围的数据自动流动，具备互联互通、即插即用、边缘网关、数据互操作、协同控制、监视与诊断等功能。
SoS级CPS	多个系统级CPS的有机组合，实现数据汇聚，对内资产优化、对外运营优化服务；功能包括数据存储、融合、分布式计算、大数据分析、数据服务等。

CPS技术体系分为**总体技术、支撑技术、核心技术**三类：

技术类型	具体内容
总体技术	系统架构、异构系统集成、安全技术、试验验证技术等（顶层设计技术）。
支撑技术	智能感知、嵌入式软件、数据库、人机交互、中间件、SDN（软件定义网络）、物联网、大数据等（应用支撑技术）。
核心技术	虚实融合控制、智能装备、MBD、数字孪生技术、现场总线、工业以太网、CAX/MES/ERP/PLM/CRM/SCM等（基础技术）。

四大核心技术要素：

- “一硬”：感知和自动控制（硬件支撑）。
- “一软”：工业软件（固化CPS计算和数据流程规则，是核心）。
- “一网”：工业网络（互联互通和数据传输的网络载体）。
- “一平台”：工业云和智能服务平台（数据汇聚和上层解决方案支撑，提供资源管控和能力服务）。

场景	说明
智能设计	产品及工艺设计、工厂设计可在虚拟空间仿真迭代；涵盖产品工艺设计、生产线/工厂设计。
智能生产	打破生产信息孤岛，实现设备互联互通、生产过程监控、资源调度优化；涵盖设备管理、生产管理、柔性制造。
智能服务	本地与远程云服务协作，实现智能装备协同优化；涵盖健康管理、智能维护、远程征兆性诊断、共享服务等。
智能应用	推动产业链转型，实现无人装备、产业链互动、价值链共赢。

建设路径分为**CPS体系设计、单元级CPS建设、系统级CPS建设、SoS级CPS建设**四个阶段。

人工智能（AI）

- **定义**：利用数字计算机或其控制的机器模拟、延伸和扩展人的智能，感知环境、获取知识并使用知识获得最佳结果的理论、方法、技术及应用系统。
- **目标**：了解智能的实质，生产出能以人类智能相似方式做出反应的智能机器；研究领域包括机器人、自然语言处理、计算机视觉、专家系统等。
- **分类**：
    - 弱人工智能：不能真正实现推理和解决问题的智能机器。
    - 强人工智能：真正能思维的智能机器。

关键技术

1. **自然语言处理（NLP）** 研究人机间自然语言有效通信的理论和方法，涵盖机器翻译（不同自然语言间的翻译）、语义理解（计算机理解文本并回答问题）、问答系统（计算机以自然语言与人交流）等。
2. **计算机视觉（CV）** 让计算机模仿人类视觉系统，具备提取、处理、理解和分析图像及图像序列的能力，将图像分析任务分解为便于管理的小块任务。
3. **知识图谱（KG）** 将不同种类的信息连接成关系网络，从“关系”角度分析问题，适用于反欺诈、不一致性验证等场景。
4. **人机交互（HCI）** 研究人与计算机之间的信息交换。
5. **虚拟现实/增强现实（VR/AR）**以计算机为核心的新型视听技术，生成与真实环境在视觉、听觉等方面高度近似的数字化环境。
6. **机器学习（ML）** 以数据为基础，通过研究样本数据寻找规律并对未来数据进行预测，广泛应用于数据挖掘、计算机视觉、自然语言处理等领域。
    1. **学习模式分类**：

类型	说明
监督学习	利用已标记的有限训练数据集建立模型，实现对新数据的标记/映射；应用于自然语言处理、垃圾邮件侦测等，算法包括回归、分类等。
无监督学习	利用无标记数据描述隐藏结构/规律；用于经济预测、异常检测等，算法包括Apriori、KMeans、随机森林等。
半监督学习	利用少量标注样本和大量未标识样本训练分类，减少标注代价；算法如图论推理、拉普拉斯支持向量机等。
强化学习	学习从环境状态到行为的映射，使智能体行为获得环境最大奖赏；应用于机器人控制、无人驾驶等，算法包括Q-Learning、时间差学习等。

2. **学习方法分类**：
   - 传统机器学习：需手动完成领域特征提取，依赖大量领域专业知识。
   - 深度学习：基于多层神经网络，无需人工特征提取，但需海量训练数据和强大算力（如GPU），注重特征学习。
3. **常见算法补充**：
   - 迁移学习：利用其他领域数据关系进行学习，适用于传感器网络定位、文字/图像分类等小规模化应用。
   - 主动学习：查询最有用的未标记样本并由专家标记，训练模型提高精度。
   - 演化学习：基于演化算法设计机器学习算法，解决复杂优化问题，应用于分类、聚类等。

机器人技术（机器人4.0时代）

机器人技术进入4.0时代，核心是**将云端大脑分布式部署**，充分利用边缘计算优势，提供高性价比服务；结合任务记忆场景的知识与常识，实现规模化部署。强调机器人需具备**感知、理解、决策能力**，可进行更加自主的服务，且需具备**更强的自适应能力**（当前服务机器人多具备物体识别、人脸识别能力，4.0时代需强化自适应）。

技术类型	说明
云-边-端的无缝协同计算	云侧提供高性能计算和知识存储，边缘侧处理数据并实现协同共享，机器人端仅完成实时操作；面向大规模机器人服务平台，信息处理分布式完成。
持续学习与协同学习	机器人通过少量数据建立基本识别能力，自主获取并自动标注更多相关数据，重新训练模型以提升性能。
知识图谱	需具备动态、个性化的知识，且需与机器人的感知、决策能力深度结合。
场景自适应	主动观察场景内人和物的变化，预测可能发生的事件，进而调整行动模式；核心是**场景预测能力**（通过细致观察场景元素，结合知识和模型分析，预测事件发生时间并改变行为）。
数据安全	需同时保障**端到端的安全传输**和**服务器端的安全存储**。

类型	说明
操作机器人	典型代表为核电站处理放射性物质时的远距离操作机器人。
程序机器人	按预先给定的程序、条件、位置进行作业。
示教再现机器人	可记录示教的操作过程，需要时重复再现；示教方法包括直接示教、遥控示教。
智能机器人	既可执行预先设定动作，也可根据工作环境变化自主变换动作。
综合机器人	由操作机器人、示教再现机器人、智能机器人组合而成（如火星机器人，可看作地面指令操纵的操作机器人）。

边缘计算

边缘计算将**数据处理、应用程序运行、功能服务实现**从网络中心下放到**网络边缘节点（靠近物或数据源的位置）**，在网络边缘侧的智能网关上就近采集并处理数据，无需将大量未处理的原生数据上传到远端大数据平台。其价值在于使海量数据就近处理、设备高效协同，理论上可满足敏捷性、实时性、数据优化、应用智能、安全与隐私保护等关键需求。

业务本质是**云计算在数据中心之外汇聚节点的延伸和演进**，核心能力发展方向为“**边云协同**”和“**边缘智能**”，包含三类落地形态：

形态类型	说明
云边缘	云服务在边缘侧的延伸，逻辑上仍属于云服务，需依赖或与云服务紧密协同。
边缘云	在边缘侧构建中小规模云服务能力，边缘服务能力主要由其提供。
云化网关	以云化技术重构嵌入式网关系统，边缘侧提供协议/接口转换、边缘计算能力；云侧控制器提供边缘节点的资源调度、应用管理与业务编排能力。

- 软件平台：需导入云理念、架构、技术，提供端到端实时、协同式智能、可信赖、可动态重置等能力。
- 硬件平台：需考虑异构计算能力。

特点	说明
联接性	是边缘计算的基础，因物理对象和应用场景的多样性，需具备丰富的联接功能。
数据第一入口	作为物理世界到数字世界的桥梁，拥有大量实时、完整的数据，可基于数据全生命周期管理创造价值，支撑预测性维护、资产效率管理等创新应用。
约束性	需适配工业现场恶劣的工作条件（如防电磁、防尘、防爆、抗振动等）；在工业互联场景下，对设备的功耗、成本、空间有较高要求。
分布性	实际部署天然具备分布式特征，需支持分布式计算与存储、分布式资源动态调度与统一管理、分布式智能、分布式安全等能力。

边云协同

- **能力分工**：云计算擅长**全局性、非实时、长周期的大数据处理与分析**（如长周期维护、业务决策支撑）；边缘计算更适用**局部性、实时、短周期的数据处理与分析**（如本地业务的实时智能化决策与执行）。
- **数据与模型闭环**：边缘是云端高价值数据的采集和初步处理单元，支撑云端应用；云端通过大数据分析优化输出的业务规则或模型，下发到边缘侧供其运行。
- **六种协同类型**：

协同类型	说明
资源协同	边缘节点提供计算、存储、网络、虚拟化等基础设施资源，具备本地资源调度管理能力，同时接受云端资源调度管理策略（含设备、资源、网络连接管理）。
数据协同	边缘节点采集现场/终端数据，初步处理后上传云端；云端提供海量数据的存储、分析与价值挖掘。
智能协同	边缘节点按AI模型执行推理，实现分布式智能；云端开展AI的集中式模型训练，并将模型下发至边缘节点。
应用管理协同	边缘节点提供应用部署与运行环境，管理本节点应用生命周期；云端提供应用开发、测试环境及全生命周期管理能力。
业务管理协同	边缘节点提供模块化、微服务化的应用/数字孪生/网络等实例；云端提供按客户需求的业务编排能力。
服务协同	边缘节点按云端策略实现部分ECSaaS服务，通过ECSaaS与云端SaaS协同实现按需SaaS服务；云端提供服务分布策略及SaaS服务能力。

数字孪生体技术

- 数字孪生体技术是**跨层级、跨尺度的现实世界和虚拟世界建立沟通的桥梁**。
- 数字孪生体是现有或将要有的**物理实体对象的数字模型**，通过实测、仿真和数据分析实时感知、诊断、预测物理实体对象的状态；通过优化和指令调控物理实体对象的行为；通过数字模型间的相互学习进化自身，同时改进利益相关方在物理实体对象生命周期内的决策。

1. **核心技术**：建模、仿真和基于数据融合的数字线程。
    - **建模**：将对物理世界的理解简化和模型化，通过数字化和模型化，用信息换能量，减少物理实体（尤其是复杂系统）的不确定性。需求指标、生存期阶段和空间尺度构成其建模技术体系的三维空间。
    - **仿真**：验证和确认对物理世界理解的正确性与有效性，是创建和运行数字孪生体、保证其与对应物理实体实现有效闭环的核心技术。通过将包含确定性规律和完整机理的模型转化为软件，模拟物理世界，若模型正确且输入信息和环境数据完整，可准确反映物理世界特性和参数。
2. **其他技术**：VR、AR、MR等增强现实技术，数字线程、系统工程和MBSE、物联网、云计算、雾计算、边缘计算、大数据技术、机器学习、区块链技术等，是数字孪生体构建过程中的内外围核心技术。

数字孪生体主要应用于**制造、产业、城市和战场**领域。

云计算

云计算的内涵包含**平台（基础设施）**和**应用**两个方面：

- 平台（基础设施）：地位相当于PC上的操作系统，是云计算应用的基础。
- 应用：所需计算与存储在“云端”完成，客户端通过互联网访问计算与存储能力。

服务类型	说明
软件即服务（SaaS）	服务商将应用软件统一部署在云计算平台上，客户通过互联网订购，服务商按订购数量、时间等收费，客户通过标准浏览器使用服务；例如在线办公软件。
平台即服务（PaaS）	服务商提供分布式开发环境与平台作为服务，客户在平台基础上定制开发应用程序；例如云开发平台。
基础设施即服务（IaaS）	服务商将多台服务器组成的“云端”基础设施作为计量服务提供，整合内存、I/O设备、存储、计算能力为虚拟资源池，为客户提供存储、虚拟化服务器等服务；例如云服务器。

服务模式特征：

- 灵活性：$\mathrm{SaaS} \rightarrow \mathrm{PaaS} \rightarrow \mathrm{IaaS}$ 依次增强。
- 方便性：$\mathrm{IaaS} \rightarrow \mathrm{PaaS} \rightarrow \mathrm{SaaS}$ 依次增强。

部署类型	说明
公有云	云基础设施公开，可自由分配给公众，企业、学术界、政府机构可拥有和管理，以低廉价格为用户提供服务；例如公共云存储服务。
社区云	云基础设施由社区组织专有，这些组织共同关注任务、安全需求、政策等，属于“公有云”范畴的一部分。
私有云	云基础设施分配给单个组织，可由该组织或第三方拥有、管理及操作；例如企业自建私有云数据中心。
混合云	公有云、私有云、社区云的组合，因安全和控制原因，企业部分信息无法放置在公有云时采用；例如企业核心数据存私有云，非核心业务用公有云。

大数据

大数据是指其大小或复杂性无法通过现有常用的软件工具，以合理的成本并在可接受的时限内对其进行捕获、管理和处理的数据集，这些困难涵盖数据的收入、存储、搜索、共享、分析和可视化等环节。具备**大规模、高速度、多样化、价值密度低、可变性、复杂性**等特点。

1. 数据获取和记录
2. 信息抽取和清洗
3. 数据集成、聚集和表示
4. 查询处理、数据建模和分析
5. 解释

AI芯片

AI芯片是一种特制的微处理器，专门为高效运行人工智能算法而设计，致力于解决AI应用中的**大规模并行计算问题**，尤其针对神经网络模型的密集型数学运算（如矩阵乘法、卷积操作、激活函数计算等）。

核心原理基于**人工神经网络**，芯片内部的处理单元**模拟生物神经元的工作机制**。每个处理单元可独立进行复杂数学运算（如权重乘以输入信号并累加，形成神经元的激活输出）；激活函数决定信号如何转化为有意义的结果，是AI芯片的关键组成部分。

类型	说明	特点
GPU（图形处理器）	原本用于图形渲染，因并行计算能力强，广泛用于训练大型深度学习模型，擅长浮点数密集型计算任务。	计算能力强，但成本高、功耗高。
FPGA（现场可编程门阵列）	具有高度灵活的可编程性，可在硬件层面快速重新配置以适应不同AI算法，适用于早期开发阶段和动态工作负载场景。	可编程、高度灵活，但计算能力不强。
ASIC（专用集成电路）	为特定AI任务定制的芯片，在特定应用中计算效率更高、能耗更低，但缺乏通用性。	体积小、功耗低、适合量产；但研发时间长、不可编辑，前期投入成本高，技术风险较大。
TPU（张量处理单元）	Google推出的ASIC实例，专门针对机器学习任务设计，专注于高效的矩阵运算，尤其适合TensorFlow框架下的深度学习模型。	高效的张量计算能力、功耗低、采用低精度计算。

补充：知识产权与标准化

标准化基础知识

分类	内容说明
国际标准（IS）	由国际标准化组织（ISO）、国际电工委员会（IEC）制定，及ISO《国际标准题内关键词索引》收录的其他国际组织标准。
国家标准（NB）	中国（GB，国家技术监督局）、美国（ANSI，国家标准协会）、英国（BS，标准学会）、日本（JIS，工业标准调查会）等国家机构制定。
区域标准	太平洋地区标准会议（PASC）、欧洲标准化委员会（CEN）、亚洲标准咨询委员会（ASAC）、非洲地区标准化组织（ARSO）等区域组织制定。
行业标准	如美国IEEE、中国GJB（国家军用标准）、美国DOD-STD（国防部标准）等行业机构制定。
企业（机构）标准	供企业内部使用的标准。
项目（课题）标准	如计算机集成制造系统（CIMS）的软件工程规范。

标准代号与编号

标准类型	编号规则
国际标准	格式为“ISO+标准号[+杠+分标准号]+冒号+发布年号”（方括号内容可选）。
国家标准	格式为“国家标准代号+标准发布顺序号-标准发布年代号（4位）”；我国强制性国标代号为GB，推荐性为GB/T。
行业标准	格式为“行业标准代号（强制或推荐）+标准发布顺序号-标准发布年代号（4位）”；如航天QJ、电子SJ等。
地方标准	格式为“DB+行政区域代码（前两位）/（强制或推荐）+地方标准发布顺序号-标准发布年代号（4位）”。
企业标准	格式为“Q/企业代号+标准发布顺序号-标准发布年代号（4位）”。

知识产权基础知识

- **定义**：民事权利主体（公民、法人）基于创造性智力成果的权利。
- **保护对象**（国际公约）：文学艺术和科学作品、表演及音像节目、发明、科学发现、工业品外观设计、商标/服务标记/商业名称、制止不正当竞争、商业秘密（世贸协议补充）等。
- **分类**：
    - 工业产权：专利、实用新型、工业品外观设计、商标、商业秘密等。
    - 著作权（版权）：
        - 人身权（精神权利）：发表权、署名权、修改权、保护作品完整性权。
        - 财产权（经济权利）：使用权、获得报酬权。

计算机软件著作权

- **法律依据**：《中华人民共和国著作权法》《计算机软件保护条例》。
- **主体资格获取途径**：
    - 公民：独立开发、委托开发（约定归己）、转让、合作开发、继承（署名权除外）。
    - 法人：组织创作、委托/转让合同、主体变更。
    - 其他组织：-**客体**：计算机程序及其有关文档。
- **受保护条件**：独立创作（非抄袭）、可被感知（思想在固定载体中表达）、逻辑合理。

知识产权保护期限

客体类型	权利类型	保护期限
公民作品/公民软件产品	署名权、修改权、保护作品完整权	无限制
公民作品/公民软件产品	发表权、使用权、获得报酬权等财产权	作者终生+死后50年（第50年12月31日）
单位作品/单位软件产品	发表权、财产权等	首次发表后50年（第50年12月31日），未发表则不保护
注册商标	-	有效期10年（期满后6个月内须续注）
发明专利权	-	20年（自申请日起）
实用新型/外观设计专利权	-	10年（自申请日起）
商业秘密	-	公开前受保护，公开后公众可用

知识产权归属判定

单位和个人的职务作品归属判定

类型	子类型	判断说明	归属
作品	职务作品	利用单位物质技术条件创作，单位承担责任	除署名权外，其他著作权归单位
作品	职务作品	合同明确约定著作权属于单位	除署名权外，其他著作权归单位
作品	职务作品	其他	作者拥有著作权，单位有权在业务范围内优先使用
软件	职务作品	属于本职工作中明确规定的开发目标	单位享受著作权
软件	职务作品	属于从事本职工作活动的结果	单位享受著作权
软件	职务作品	使用单位资金、专用设备、未公开信息等物质技术条件，单位承担责任	单位享受著作权
专利权	职务作品	本职工作中作出的发明创造	单位享受专利权
专利权	职务作品	履行本单位交付的本职工作之外的任务所作出的发明创造	单位享受专利权
专利权	职务作品	离职、退休或调动工作后一年内，与原单位工作相关	单位享受专利权

委托、合作及商标专利归属判定

类型	子类型	判断说明	归属
作品/软件	委托创作	合同中明确约定著作权归属委托方	委托方
作品/软件	委托创作	合同中未约定著作权归属	创作方
作品/软件	合作开发	只进行组织、提供咨询意见、物质条件等辅助工作	不享有著作权
作品/软件	合作开发	共同创作的	共同享有，按人头比例；成果可分割的，可分开申请
商标	-	谁先申请谁拥有；同时申请，谁先使用谁拥有（需提供证据）；无法提供证据，协商归属，协商无效抽签决定	按上述规则判定
专利	-	谁先申请谁拥有；同时申请则协商归属，协商不成的，该发明成为社会共有技术	按上述规则判定

计算机软件著作权的侵权与非侵权判定

侵权的判定

1. 未经著作权人同意而发表或登记其软件作品。
2. 将他人软件当作自己的作品发表或登记。
3. 未经合作者同意将共同开发的软件当作自己的作品发表或登记。
4. 在他人开发的软件上署名或更改他人署名。
5. 未经著作权人或其合法受让者许可，实施修改、翻译、复制或部分复制、向公众发行或出租、办理权利许可或转让或通过网络传播其作品的行为。

非侵权的判定：为了学习和研究软件内含的设计思想和原理，通过安装、显示、传输或存储软件的方式使用软件时，可以不经许可，不支付报酬。

数学和经济管理（暂略）

案例-Web系统设计

Web应用技术分类

维度	涉及技术内容
从架构来看	MVC、MVP、MVVM、REST、Webservice、微服务
从缓存来看	MemCache、Redis、Squid
从并发分流来看	集群（负载均衡）、CDN
从数据库来看	主从库（主从复制）、内存数据库、反规范化技术、NoSQL、分区（分表）技术、视图与物化视图
从持久化来看	Hibernate、Mybatis
从分布存储来看	Hadoop、FastDFS、区块链
从数据编码看	XML、JSON
从Web应用服务器来看	Apache、Tomcat、JBOSS、IIS、WebSphere、Weblogic
其它	有状态与无状态、响应式Web设计等

Web技术演变阶段

1. 单台机器到数据库与Web服务器分离
    - **初始架构**：Web应用与数据库部署在同一台机器，资源竞争严重，性能瓶颈明显。
    - **演变后架构**：将数据库独立为**数据库服务器**，Web应用部署在**应用服务器**，实现资源分离，提升系统可维护性与性能。
2. 应用服务器集群
    - **面临问题**：单台应用服务器存在性能瓶颈，需解决“用户请求由谁转发到具体服务器”“用户访问不同服务器时Session一致性如何维护”（负载均衡与有状态/无状态问题）。
    - **演变后架构**：多台应用服务器组成**应用服务器集群**，共同连接数据库服务器，为后续负载均衡技术引入奠定基础。
3. 采用负载均衡技术
    - **演变后架构**：在客户端与应用服务器集群之间引入**负载均衡器**，负责分发用户请求到不同应用服务器；客户端通过Cookie携带Session，保障会话一致性，实现请求分流与负载均衡。
4. 数据库集群（主从库）
    - **演变后架构**：数据库分为**主库**（负责写操作）和**从库**（负责读操作），主从库同步数据，分担数据库读写压力，提升数据库性能与可用性。
5. 用缓存缓解数据库读写压力
    - **演变后架构**：引入**缓存集群**（如Memcache），应用服务器读取数据时优先从缓存获取，减少对数据库的直接访问，缓解数据库读写压力，提升系统响应速度。

CDN（内容分发网络）

CDN（Content Delivery Network）是构建在网络之上的内容分发网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。其关键技术主要为内容存储和分发技术。

基本原理：广泛采用各种缓存服务器，将其分布到用户访问相对集中的地区或网络中；在用户访问网站时，利用全局负载技术将用户访问指向距离最近的正常缓存服务器，由缓存服务器直接响应用户请求。主要加速静态资源，如HTML、CSS、JS、图片、视频等。

REST（表述性状态转移）

REST（Representational State Transfer）是一种针对网络应用设计和开发的架构风格，可降低开发复杂性，提高系统可伸缩性，目的是让不同软件或应用程序在任何网络环境下都能进行信息互相传递。**RESTful**是遵循REST原则的Web服务，是REST的形容词。

- **核心思想**：将Web应用程序的功能作为资源来表示，使用统一资源标识符（URI）对资源进行操作，并通过HTTP协议（GET、POST、PUT、DELETE等）定义对资源的操作，强调无状态、缓存机制、统一接口、分层系统、客户端-服务器分离等原则。
- **核心概念**：
    - **资源**：以资源为中心构建，互联网中一切暴露给客户端的事物都可视为资源，借助URI标识Web上的资源，资源和URI是一对多关系。
    - **表述**：描述资源在Web中某一时刻的状态，客户端和服务端通过RESTful API传递数据实现资源表述交互，常用表现形式有HTML、JSON、XML、纯文本等，资源表述返回格式需统一。
    - **状态转移**：REST定义的状态分为应用状态（客户端维护，可降低服务端并发请求压力）和资源状态（服务端保存，保证同资源请求表述一致），状态转移借助HTTP方法（如GET、POST、DELETE）实现。
    - **超链接**：通过在页面中嵌入链接与其他资源建立联系，在资源表述中添加相关资源URI，将资源接口暴露给客户端，便于实现资源状态转移，超链接由客户端维护保存。
    - 补充：REST是一种**设计风格而非架构**。
- **主要特点**：
    - **无状态性**：服务器不保存客户端状态信息，每次请求独立，便于构建可伸缩服务器。
    - **缓存**：允许客户端缓存GET请求的响应，显著提高应用程序性能。
    - **统一接口**：客户端和服务器通过统一接口交互，使用标准HTTP方法表示对资源的操作，用标准HTTP状态码表示请求结果。
    - **分层系统**：客户端通过中间层（如负载均衡器、安全层）与服务器通信，中间层对客户端透明。
    - **客户端-服务器**：基于客户端-服务器模型，二者松耦合，利于系统扩展和维护。

微服务

微服务架构将一个大型的单个应用或服务划分成一组**微型、可独立部署的服务**，围绕业务领域拆分服务，每个服务可独立开发、管理和迭代，彼此通过统一接口交流，实现分散组件的部署、管理与服务功能，简化产品交付，达到有效拆分应用、实现敏捷开发与部署的目的。

优势类型	说明
复杂应用解耦	将单一模块应用分解为多个微服务，保持总体功能不变。
独立开发与部署	每个微服务可独立开发、部署，具备独立运行进程。
技术选型灵活	开发团队可根据业务需求选择合适的体系架构与技术。
容错性强	微服务相互独立，故障被隔离在单个服务中，其他服务可通过重试、平稳退化等机制实现应用层容错。
松耦合、易扩展	服务间松耦合，可根据实际需求独立扩展，体现架构灵活性。

挑战类型	说明
系统适配性限制	并非所有系统都能转成微服务，例如数据库层的底层操作不推荐服务化。
部署复杂度提升	系统由众多微服务搭建，每个微服务需单独部署，容器技术可解决此问题。
性能问题	服务间通信依赖标准接口，可能产生延迟或调用出错（如频繁数据访问会带来较大延迟）。
数据一致性问题	分布式部署的微服务在保持数据一致性方面比传统架构更困难。

XML（可扩展标记语言）

是一种用于标记电子文件以使其具有结构性的标记语言，可标记数据、定义数据类型，允许用户自定义标记语言。

- **优点**：
    - 格式统一，符合标准；
    - 便于与其他系统远程交互，数据共享方便。
- **缺点**：
    - 文件庞大、格式复杂，传输占用带宽；
    - 服务器端和客户端解析需大量代码，代码复杂且不易维护；
    - 不同浏览器解析方式不一致，需重复编写代码；
    - 解析花费较多资源和时间。

JSON（JavaScript Object Notation）

是一种轻量级的数据交换格式，具有良好的可读性和编写便利性，可在不同平台间进行数据交换。

- **优点**：
    - 数据格式简单，易于读写，格式压缩，占用带宽小；

    - 易于解析，客户端JavaScript可通过`eval()`快速读取；
    - 支持多种编程语言（如ActionScript、C、Java等），便于服务器端解析；
    - 大幅简化服务器端和客户端的代码开发量，且易于维护。
- **缺点**：通用性不如XML广泛。

其他技术概念

类型	说明
无状态服务	处理单次请求不依赖其他请求，请求所需信息包含在请求内或可从外部（如数据库）获取，服务器本身不存储信息。
有状态服务	会在自身保存数据，先后请求存在关联。

响应式Web设计：一种可根据用户行为和设备环境智能调整布局的网络页面设计布局，以提供最佳显示效果。

- **方法与策略**：
    - 采用流式布局和弹性化设计：使用相对单位（如百分比）设置页面元素大小；
    - 响应式图片：同比缩放图片，且在小设备上降低图片分辨率。

例题一

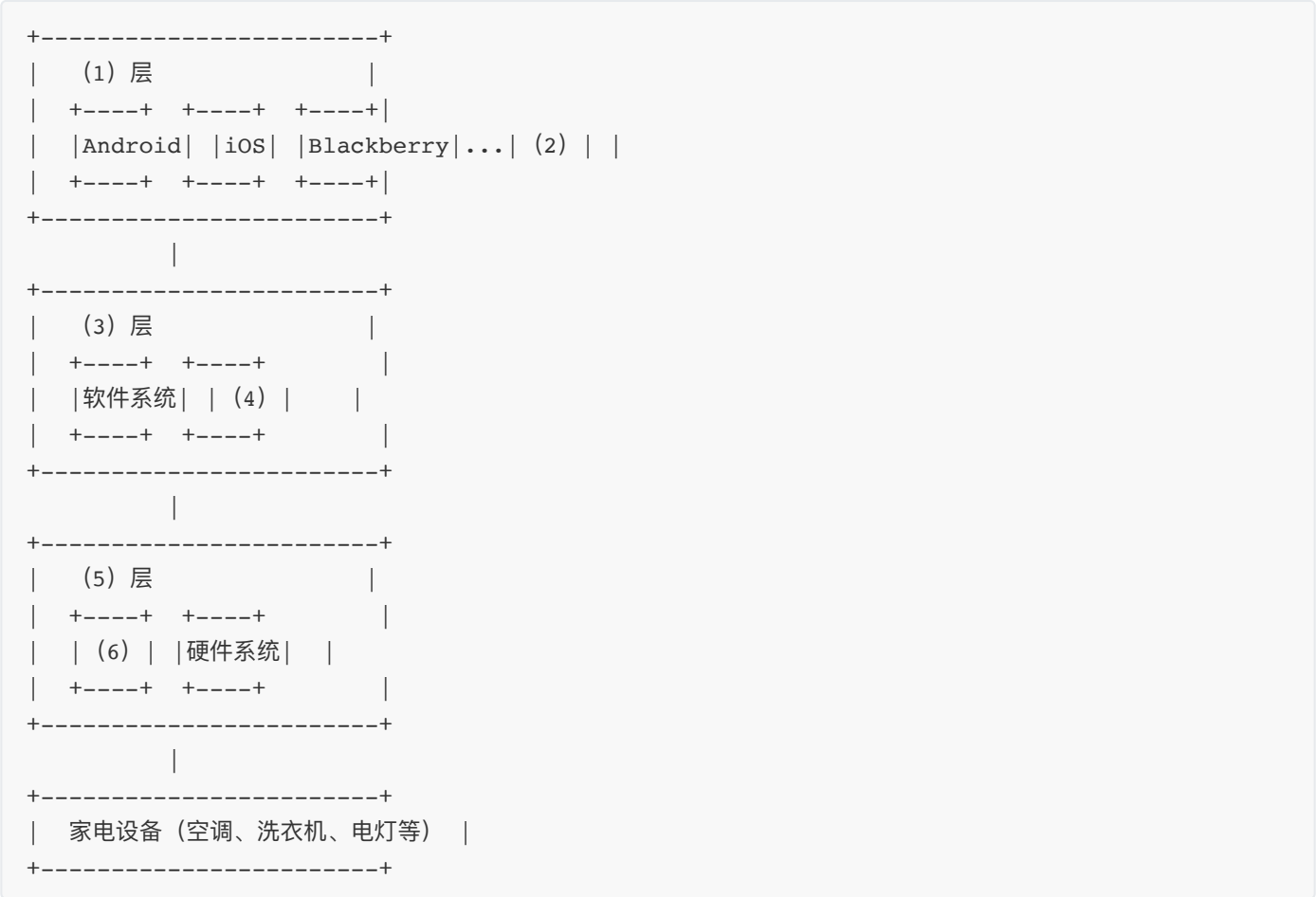
【题干】
某公司拟开发一个智能家居管理系统，该系统的主要功能需求如下：
1）用户可使用该系统客户端实现对家居设备的控制，且家居设备可向客户端反馈实时状态；
2）支持家居设备数据的实时存储和查询；
3）基于用户数据，挖掘用户生活习惯，向用户提供家居设备智能化使用建议。
基于上述需求，该公司组建了项目组，在项目会议上，张工给出了基于家庭网关的传统智能家居管理系统的设计思路，李工给出了基于云平台的智能家居系统的设计思路。经过深入讨论，公司决定采用李工的设计思路。

【问题1】
请用400字以内的文字简要描述基于家庭网关的传统智能家居管理系统和基于云平台的智能家居管理系统在网关管理、数据处理和系统性能等方面的特点，以说明项目组选择李工设计思路的原因。

【问题1 解答】

- **网关管理**：基于云平台的智能家居系统可将分散的智能家居网关数据集中起来，实现对智能家居网关的**远程高效管理**；传统家庭网关仅能在本地进行管理，分散且缺乏远程管理能力。
- **数据处理**：云端服务器可以对智能家居数据进行备份存储，当家庭网关由于故障等原因导致数据丢失时，可通过云端管理系统对网关数据进行恢复，从而提高数据的容灾备份能力；传统家庭网关数据仅本地存储，故障易造成数据丢失，容灾性弱。
- **系统性能**：基于云服务平台的智能家居管理系统将数据信息存储在云端，减少了数据请求时间，提高了通信效率；传统家庭网关本地存储数据，访问延迟较高。
  综上，云平台方案在网关管理、数据安全、系统性能上更具优势，因此项目组选择李工的设计思路。

【问题2】
请参考以下字符画形式的系统架构图，从选项（a）~（j）中选择合适内容，补充完善空（1）~（6）处的内容，协助李工完成该系统的架构设计方案。



选项：
（a）Wi-Fi
（b）蓝牙
（c）驱动程序
（d）数据库
（e）家庭网关
（f）云平台
（g）微服务
（h）用户终端
（i）鸿蒙
（j）TCP/IP

【问题2 解答】

空号	答案	解析
(1)	（h）用户终端	该层包含Android、iOS等用户终端系统，是用户与系统交互的入口层。
(2)	（i）鸿蒙	属于用户终端的操作系统之一，与Android、iOS、Blackberry等并列。
(3)	（f）云平台	是系统的核心层，承担软件系统运行、数据存储等核心服务。
(4)	（d）数据库	用于存储智能家居系统的各类数据，支撑数据的实时存储和查询需求。
(5)	（e）家庭网关	连接家电设备与云平台，是硬件系统的上层组件，负责设备接入与数据转发。
(6)	（c）驱动程序	用于适配家电设备的硬件接口，属于家庭网关的软件支撑层，保障设备与系统的通信。

【问题3】
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。

【问题3 解答】

- **TCP协议**：在IP协议提供的不可靠数据服务基础上，采用重发技术，为应用程序提供**可靠的、面向连接的、全双工**的数据传输服务。适用于传输数据量较少且对可靠性要求高的场合（如家居设备控制）。
- **UDP协议**：是**不可靠、无连接**的协议，错误检测功能较弱，仅保证应用程序进程间的通信。适用于对可靠性要求低、数据量大的场景。
- **协议选择**：该系统需实现用户终端与服务端的双向可靠通信，且家居设备控制传输数据量较少，因此应采用**TCP协议**。

例题二

某公司拟开发一款基于Web的工业设备监测系统，以实现对多种工业设备数据的分类采集、运行状态监测以及相关信息的管理。该系统应具备以下功能：

- 现场设备状态采集功能：根据数据类型对设备监测指标状态信号进行分类采集；
- 设备采集数据传输功能：利用可靠的传输技术，实现将设备数据从制造现场传输到系统后台；
- 设备监测显示功能：对设备的运行状态、工作状态以及报警状态进行监测并提供相应的图形化显示界面；
- 设备信息管理功能：支持设备运行历史状态、报警记录、参数信息的查询。

同时，该系统还需满足以下非功能性需求：
（a）系统应支持大于100个工业设备的并行监测；
（b）设备数据从制造现场传输到系统后台的传输时间小于1s；
（c）系统应7×24小时工作；
（d）可抵御常见XSS攻击；
（e）系统在故障情况下，应在0.5小时内恢复；
（f）支持数据审计。

面对系统需求，公司召开项目组讨论会议，制定系统设计方案，最终决定采用三层拓扑结构，即现场设备数据采集层、Web监测服务层和前端Web显示层。

【问题1】
请按照**性能、安全性和可用性**三类非功能性需求分类，选择题干描述的（a）～（f）填入（1）～（3）。

表5-1 非功能性需求归类表

非功能性需求类别	非功能性需求
性能	（1）
安全性	（2）
可用性	（3）

【问题1 解答】

非功能性需求类别	非功能性需求	解析
性能	（1）a、b	a支持并行监测、b传输时间小于1s，均属于性能需求。
安全性	（2）d、f	d抵御XSS攻击、f支持数据审计，均属于安全性需求。
可用性	（3）c、e	c7×24小时工作、e故障恢复时间，均属于可用性需求。

【问题2】
该系统的Web监测服务层拟采用SSM开发。SSM框架的工作流程图如图5-1所示，完善图5-1中（1）～（7）处空白的内容。

```
+----------------+  +--------+
| HTML/CSS/JS    |  | (7)    |
+----------------+  +--------+
          |            |
          v   ---------|
+----------------+
| JSP Engine     |  <--- (6)
+----------------+
          |
          v
+----------------+
| Spring MVC     |  <-- Business Layer POJO
+----------------+
          |
          v
+----------------+
| (4)            |  <--- (5)
+----------------+
          |
          v
+----------------+
| (3)            |  <--- (2)
+----------------+
          |         <--- (1)
```

```
          v
+----------------+
| Database       |
+----------------+
```

选项：
（a）Connection Pool
（b）Struts2
（c）Persistent Layer
（d）Mybatis
（e）HTTP
（f）MVC
（g）Kafka
（h）View Layer
（i）JSP
（j）Controller Layer
（k）Spring

【问题2 解答】

空号	答案	解析
（1）	（a）Connection Pool	用于数据库连接管理，是数据库访问的基础组件。
（2）	（c）Persistent Layer	对应持久层，负责数据持久化操作。
（3）	（d）Mybatis	SSM框架中的持久层框架，实现数据库交互。
（4）	（k）Spring	核心框架，管理业务逻辑与依赖注入。
（5）	（j）Controller Layer	Spring MVC的控制层，处理请求与响应。
（6）	（h）View Layer	视图层，负责页面展示逻辑。
（7）	（i）JSP	Java服务器页面，属于视图层技术实现。

【问题3】
该工业设备检测系统拟采用工业控制领域中统一的数据访问机制，实现与多种不同设备的数据交互，请用200字以内的文字说明采用标准的数据访问机制的原因。

【问题3 解答】
标准的数据访问机制可在硬件供应商与软件开发商间建立统一规则，屏蔽不同通信协议的差异，为上层应用提供统一访问接口，实现不同总线协议设备的互操作，使开发脱离底层细节。它独立于平台，支持多厂商设备信息无缝传输，具备语言无关性、代码重用性、易于集成性等优点。硬件升级或修改时仅需改动硬件接口，不影响上层应用（如工业自动化领域常用的OPC协议）。

案例-嵌入式系统

例题一

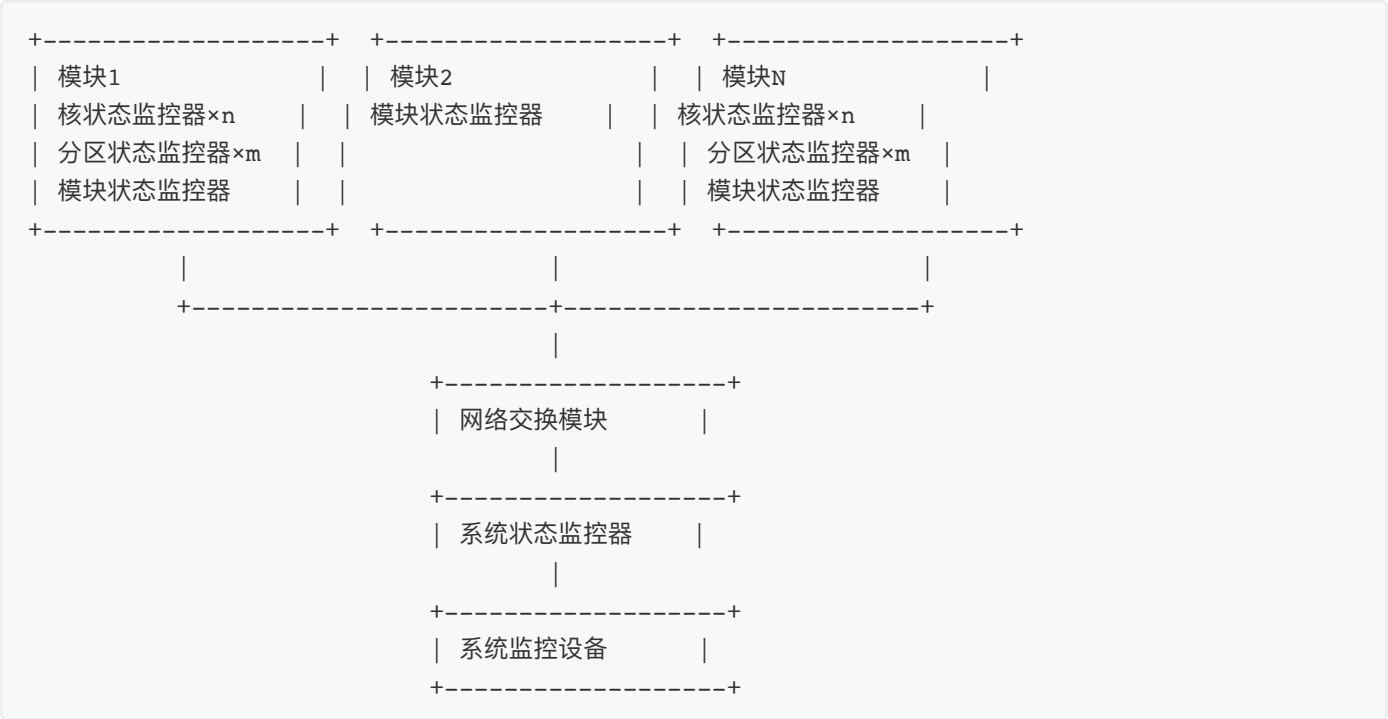
【题干】
系统的故障检测和诊断是宇航系统提高装备可靠性的主要技术之一。随着装备信息化发展，分布式架构下资源配置增多、布局分散，对系统故障检测和诊断方法提出新要求。为适应宇航装备分布式综合化电子系统发展，解决因资源部署分散导致的系统状态综合与监控困难问题，公司安排张工研究，张工提出针对该架构的故障检测和诊断方案。

【问题1】
张工提出：宇航装备软件架构采用四层层次化体系结构（模块支持层、操作系统层、分布式中间件层、功能应用层）。为实现分布式系统故障检测和诊断能力，方案建议将其构建在分布式中间件内，通过心跳或超时探测技术实现故障检测器。请用300字以内文字分别说明心跳检测和超时探测技术的基本原理及特点。

【问题1 解答】

- **心跳检测技术**：节点以固定频率向其他节点发送心跳信息表示存活。若一段时间未收到某节点心跳，判定该节点失效，其资源和服务会被接管。**特点**：响应速度快，但易产生误判。
- **超时探测技术**：探测节点向被探测节点发送PING信号，被探测节点收到后回复ECHO信号并附带状态信息。若预定时间内未收到ECHO信号，判定被探测节点失效。**特点**：可获得详细探测结果，但判断周期较长。

【问题2】
张工针对分布式综合化电子系统架构特征，给出初步设计方案：每个节点的故障监测与诊断器监控系统所有故障信息，经综合分析判断后，由故障诊断器分析原因并给出解决方案。系统为处理机核配置**核状态监控器**、为分区配置**分区状态监控器**、为模块配置**模块状态监控器**、为系统配置**系统状态监控器**（架构如字符画所示）。



请根据故障(a)-(h)，判断其所属监控器检测范围，完善表3-1。
故障列表：
(a)应用程序除零
(b)看门狗故障
(c)任务超时
(d)网络诊断故障
(e)BIT检测故障
(f)分区堆栈溢出

(g)操作系统异常
(h)模块掉电

表3-1 故障分类

监控器类型	故障项
核状态监控器	(1)、(2)
分区状态监控器	(3)
模块状态监控器	(4)、(5)、(6)
系统状态监控器	(7)、(8)

【问题2 解答】

监控器类型	故障项	解析
核状态监控器	(1)b、(2)e	看门狗故障（b）、BIT检测故障（e）属于核级故障，由核状态监控器检测。
分区状态监控器	(3)f	分区堆栈溢出（f）属于分区级故障，由分区状态监控器检测。
模块状态监控器	(4)a、(5)d、(6)h	应用程序除零（a）、网络诊断故障（d）、模块掉电（h）属于模块级故障，由模块状态监控器检测。
系统状态监控器	(7)g、(8)c	操作系统异常（g）、任务超时（c）属于系统级故障，由系统状态监控器检测。

【问题3】
张工提出系统故障诊断采用故障诊断器，可综合故障信息和系统状态，依据智能决策数据库策略判定故障类型和处理方法。智能决策数据库对故障可开展定性或定量分析，定量分析常用基于解析模型的方法和数据驱动的方法。张工提议采用基于解析模型的方法，王工反对并认为数据驱动方法更适合分布式综合化电子系统架构。请用300字以内文字说明数据驱动方法的基本概念，以及王工提出采用此方法的理由。

【问题3 解答】

- **数据驱动方法基本概念**：从初始数据或观测值出发，运用启发式规则寻找并建立内部特征关系，发现定理或定律的问题求解方法，也指基于大规模统计数据的方法（如机器学习、统计分析法、信号分析法）。
- **王工采用此方法的理由**：分布式综合化电子系统环境复杂，宇航系统难以精确建立预定制的解析模型；数据驱动方法可通过已有数据训练模型，实现模型逐渐精细化，且兼容未来系统变化，更适配该架构的故障定量分析需求。

例题二

【题干】
某软件公司开发一项基于数据流的软件，其系统主要功能是对输入数据进行多次分析、处理和加工，生成输出数据。需求方对系统软件可靠性要求很高，要求长时间无故障运行。公司将系统设计交给王工负责，王工给出系统模块示意图（图5-1，串联结构），并解释"只要各个模块可靠度足够高，失效率足够低，整个软件系统可靠性有保证"。

李工对王工方案提出异议：第一，若各模块可靠度均为0.99，系统串联则可靠度为$0.99^4 \approx 0.96$，下降明显；第二，串联结构中一个模块失效则整个系统失效。

李工认为应采用冗余技术中的动态冗余或N版本程序设计技术，对易失效或重要模块进行冗余设计，将部分串联结构变为并联结构（图5-2）。

刘工建议，李工方案中M1和M4未采用容错设计，若其故障可能导致严重后果，可在M1和M4设计上采用检错技术，故障后及时发现并报警。

注：假设各模块可靠度均为0.99。

图5-1 系统模块示意图（串联）

```
输入 → M1 → M2 → M3 → M4 → 输出
```

图5-2 系统模块示意图（动态冗余后）

```
输入 → M1 → [M2-1  M2-2  M2-3] → [M3-1  M3-2  M3-3] → M4 → 输出
           （并联）                （并联）
```

【问题1】（4分）
在系统可靠性中，可靠度和失效率是两个关键指标，请分别解释其含义。

【问题1 解答】

- **可靠度**：系统在规定条件下、规定时间内不发生失效的概率。
- **失效率**：又称风险函数或条件失效强度，指运行至此刻系统未失效的情况下，单位时间软件系统出现失效的概率。

【问题2】（13分）
请解释李工提出的动态冗余和N版本程序设计技术，给出图5-1中模块M2采用图5-2动态冗余技术后的可靠度。请给出采用李工设计方案后整个系统可靠度的计算方法，并计算结果。

【问题2 解答】

- **动态冗余**：又称主动冗余，通过故障检测、定位及恢复等手段实现容错。主要方式是多重模块待机储备，当工作模块出错时，用备用模块替代并重新运行。备用模块待机时可与主模块同工作（热备份系统，如双重系统）或不工作（冷备份系统，如双工系统、双份系统）。
- **N版本程序设计**：一种静态故障屏蔽技术，用N个具有相同功能的程序同时执行计算，结果通过多数表决选择。N个版本需由不同人员独立设计，使用不同方法、语言、开发环境和工具，以减少表决点上相关错误的概率。
- **M2动态冗余后的可靠度**：$R_{M2} = 1 - (1 - 0.99)^3 = 0.999999$
- **整个系统可靠度计算方法及结果**：
  李工方案中，M1、M4为串联，M2、M3为并联。

系统可靠度 = $R_{M1} \times R_{M2冗余} \times R_{M3冗余} \times R_{M4}$
代入数值：$0.99 \times 0.999999 \times 0.999999 \times 0.99 \approx 0.98$

【问题3】（8分）
请给出检错技术的优缺点，并说明检测技术常见的实现方式和处理方式。

【问题3 解答】

- **优缺点**：
  - 优点：实现代价一般低于容错技术和冗余技术。
  - 缺点：不能自动解决故障，若不人工干预，最终会导致软件系统无法正常运行。
- **常见实现方式**：
  - 判断返回结果：若返回结果超出正常范围，进行异常处理。
  - 计算运行时间：若模块或函数运行时间超过预期，判断出现故障。
  - 置状态标志位等：根据实际情况选用自检实现方式。
- **处理方式**：
  大多数采用“查出故障-停止软件系统运行-报警”的处理方式；也可根据故障是否需实时处理，采用不停止或部分停止软件系统运行的情况。

案例-软件架构

软件架构风格常考点

分类及核心特点

架构风格	子风格	常考关键字及实例	简介
数据流	批处理	-	数据以**整体方式**传递，一个接一个处理。
数据流	管道-过滤器	传统编译器、网络报文处理	前一个输出是后一个输入，构件相对独立。
调用/返回	主程序/子程序	-	显示调用，主程序直接调用子程序。
调用/返回	面向对象	-	构件是对象，通过对象调用封装的方法和属性。
调用/返回	层次型	-	分层结构，每层最多影响上下两层，存在调用关系。
调用/返回	C/S	瘦客户机	三层结构：客户端、应用服务器、数据库服务器。
调用/返回	B/S	零客户端	三层结构：浏览器、Web服务器、数据库服务器。
以数据为中心	仓库	现代编译器IDE	中央共享数据源，独立处理单元。
以数据为中心	黑板	信号处理、编译器优化	包含知识源、黑板、控制三部分，解决复杂非结构化问题。
虚拟机	解释器	自定义规则、跨平台适配	解释自定义规则，含解释引擎、存储区、数据结构。
虚拟机	规则系统	-	含规则集、解释器、选择器及工作内存，适用于人工智能、DSS。
独立构件	进程通信	-	进程间消息传递，点对点独立传递（同步/异步）。
独立构件	事件系统	程序语法高亮、语法错误提示	基于事件的隐式调用，通过事件触发调用。
C2风格	-	构件和连接件、顶部和底部	由连接件绑定的并行构件网络，按一组规则运作。

常考架构风格对比

架构风格	主要特点	主要优点	主要缺点	适合领域
管道-过滤器	过滤器相对独立	高内聚低耦合、支持复用、可维护性/扩展性强、并发灵活	不适于交互性强的应用、数据流关系需协调	模块清晰独立、接口明确的系统
解释器风格	系统核心是虚拟机	自定义规则、跨平台适配、可扩展性强	执行效率低、规则过多则复杂度高、性能差	模式匹配系统、语言编译器
面向对象	构件是对象，通过函数/过程调用交互	高度模块化、封装性好、代码共享灵活、易维护、性能好	增加对象间依赖关系	多领域通用
事件系统	构件触发/广播事件而非直接调用过程	支持复用、易实现并发多任务、可扩展、简化代码	对系统计算控制能力弱、对象逻辑关系复杂	可通过事件处理表征外部表现的系统
层次型	构件分层，每层为上下层提供服务	支持逐级抽象、可扩展、支持复用	耦合度高的系统难划分层次	功能层次抽象、低耦合系统
仓库	中央数据结构+独立构件操作数据	数据集中、以数据为中心	仅适合特定领域	以数据为中心的系统

MVC架构

核心模块

- **模型（Model）**：处理应用程序**数据逻辑**，负责在数据库中存取数据，代表业务数据和业务逻辑。
- **视图（View）**：处理**数据显示**，依据模型数据创建，是用户可见并交互的界面（仅显示/接收输入，不处理业务逻辑）。
- **控制器（Controller）**：处理**用户交互**，从视图读取数据、控制用户输入，并向模型发送数据。

优势

- 便于管理复杂应用：可在某一时刻专注一个模块（如不依赖业务逻辑，仅专注视图设计）。
- 测试更高效：分层结构使各模块测试更具针对性。
- 简化团队开发：不同开发人员可同时并行开发视图、控制器逻辑和业务逻辑。

- 增强系统扩展性、强壮性与灵活性：业务处理与显示分离，适配主流Web应用框架（如Spring MVC等）。

J2EE架构

层次	组件及说明
客户层组件	可基于Web方式（如静态HTML页面、Applets）或传统方式，是用户与系统交互的入口。
Web层组件	由JSP页面或Servlet组成，负责处理Web请求，衔接客户层与业务层。
业务层组件	包含特定领域的业务逻辑处理，是系统核心业务逻辑的实现层。
信息系统层	处理企业级信息系统软件（如ERP、大型机事务处理、数据库系统等），为J2EE应用提供底层数据和系统支持。

JSP+Servlet+JavaBean+DAO的MVC实现

组件	角色（MVC对应）	功能说明
JSP	视图（V）	负责显示、收集数据，是用户可见的交互界面。
Servlet	控制器（C）	属于业务逻辑层，处理复杂业务逻辑（如数据验证、实例化JavaBean、调用DAO操作数据库等），调用Service方法处理服务。
JavaBean	模型（M）组成部分	用于数据封装，在Servlet和JSP之间传递查询结果等数据。
DAO	模型（M）组成部分	负责连接数据库并执行数据库操作（如查询、删除、更改等）。

- **模型（M）**：由DAO和JavaBean共同组成，处理数据逻辑和持久化。
- **基本流程**：JSP将数据发送给Servlet → Servlet解析数据并调用相应Service → 若需数据库交互，Service通过DAO操作数据库，用JavaBean封装结果 → 结果返回给Servlet → Servlet再返回给JSP展示。

面向服务的架构（SOA）

一种设计理念，包含多个**独立部署运行**的服务，服务之间通过网络调用，相互依赖最终提供一系列完整功能。

企业服务总线（ESB）是连接各个服务节点的“管道”，用于集成基于不同协议的服务，通过**消息转化、解释及路由**，实现不同服务的互联互通。

特点序号	内容说明
1	SOA的一种实现方式，在SOA中起**总线作用**，连接并整合各种服务。
2	描述服务的元数据，支持服务注册管理。
3	在服务请求者与提供者之间传递并转换数据，支持同步、异步等交互模式。
4	具备**发现、路由、匹配和选择能力**，解耦服务请求者与提供者；还支持安全、服务质量保证、负载平衡等高级能力。

主要功能：服务位置透明性，传输协议转换，消息格式转换，消息路由，消息增强，安全性，监控与管理。

例题一

【题干】
某电子商务公司拟升级其会员与促销管理系统，以向用户提供个性化服务、提高用户粘性。项目主要目标是提升会员管理方式的灵活性，因当前用户规模不大、业务相对简单，系统性能方面不做过多考虑。新系统除保持现有四级固定会员制度外，还需根据用户消费金额、偏好、重复性等特征动态调整商品折扣力度，并支持在特定活动周期内主动筛选与活动主题高度相关的用户集合，提供个性化打折促销活动。

系统需求和质量属性描述如下：（选择对应条目完善质量效用树，略）

系统架构师给出两种候选架构设计方案，公司组织专家对系统架构进行评估。

【问题1】（12分）

在架构评估过程中，质量属性效用树（utility tree）是对系统质量属性进行识别和优先级排序的重要工具。请将合适的质量属性名称填入图1-1中(1)、(2)空白处，并选择题干描述的(a)~(k)填入(3)~(6)空白处，完成该系统的效用树。

牢记：性能、安全、可用性、可修改性

【问题2】（13分）

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（interpreters）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。请针对系统的主要功能，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。

【问题2 解答】
该系统更适合采用**解释器架构风格**，分析如下：

比较维度	面向对象架构风格	解释器架构风格
折扣规则的可修改性	规则封装在对象中，修改需调整类结构，灵活性低。	规则是独立语法规则，解释器可直接解析变化的规则，修改更便捷。
个性化折扣定义灵活性	需通过对象方法调整，难以快速适配“千人千面”的个性化需求。	可根据用户特征灵活解释执行规则，能高效实现个性化折扣定义。
系统性能	实现相对固定，执行效率高。	运行期动态绑定执行，性能略差。

由于项目核心目标是**提升灵活性**，且业务简单、系统性能要求不高，因此**解释器架构风格**更适配需求。

例子二

【题干】
某公司拟开发一套机器学习应用开发平台，支持用户使用浏览器在线进行基于机器学习的智能应用开发活动。该平台的核心应用场景是用户通过拖拽算法组件灵活定义机器学习流程，采用自助方式进行智能应用设计、实现与部署，并可以开发新算法组件加入平台中。

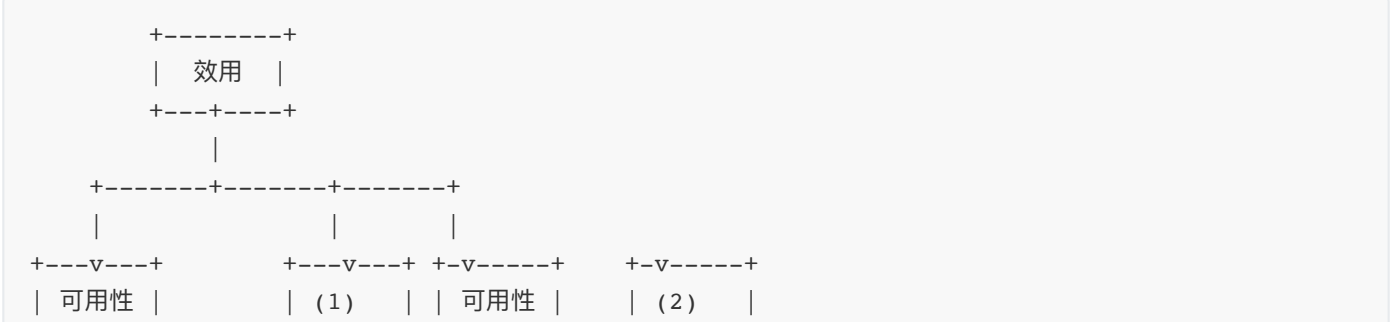
系统需求和质量属性描述如下：
(a)平台用户分为算法工程师、软件工程师和管理员等三种角色，不同角色的功能界面有所不同；
(b)平台应该具备数据库保护措施，能够预防核心数据库被非授权用户访问；
(c)平台支持分布式部署，当主站点断电后，应在20秒内将请求重定向到备用站点；
(d)平台支持初学者和高级用户两种界面操作模式，用户可以根据自己的情况灵活选择合适的模式；
(e)平台主站点宕机后，需要在15秒内发现错误并启用备用系统；
(f)在正常负载情况下，机器学习流程从提交到开始执行，时间间隔不大于5秒；
(g)平台支持硬件扩容与升级，能够在3人天内完成所有部署与测试工作；
(h)平台需要对用户的所有操作过程进行详细记录，便于审计工作；
(i)平台部署后，针对界面风格的修改需要在3人天内完成；
(j)在正常负载情况下，平台应在0.5秒内对用户的界面操作请求进行响应；
(k)平台应该与目前国内外主流的机器学习应用开发平台的界面风格保持一致；
(l)平台提供机器学习算法的远程调试功能，支持算法工程师进行远程调试。

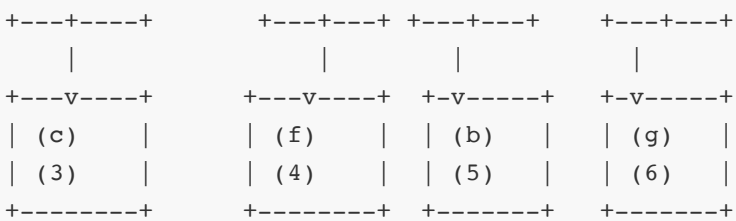
公司架构师给出三种候选架构设计方案，组织专家对平台架构进行评估。

【问题1】

在架构评估过程中，质量属性效用树（utility tree）是对系统质量属性进行识别和优先级排序的重要工具。请将合适的质量属性名称填入图1-1中(1)、(2)空白处，并从题干中的(a)-(l)中选择合适的质量属性描述，填入(3)-(6)空白处，完成该平台的效用树。

图1-1 机器学习应用开发平台效用树（字符画形式）





【问题1 解答】

空白处	答案	解析
(1)	性能	对应需求(f)（机器学习流程执行时间）和(j)（界面操作响应时间），属于性能质量属性。
(2)	安全性	对应需求(b)（数据库保护）和(h)（用户操作审计），属于安全性质量属性。
(3)	e	需求(e)（主站点宕机后备用系统启用）属于可用性维度。
(4)	j	需求(j)（界面操作响应时间）属于性能维度。
(5)	h	需求(h)（用户操作审计）属于安全性维度。
(6)	i	需求(i)（界面风格修改时间）属于可修改性维度。

【问题2】

针对该系统的功能，赵工建议采用解释器（interpreter）架构风格，李工建议采用管道-过滤器（pipe-and-filter）的架构风格，王工则建议采用隐式调用（implicit invocation）架构风格。请针对平台的核心应用场景，从机器学习流程定义的灵活性和学习算法的可扩展性两个方面对三种架构风格进行对比与分析，并指出该平台更适合采用哪种架构风格。

【问题2 解答】

该平台更适合采用**解释器架构风格**，分析如下：

比较维度	解释器架构风格	管道-过滤器架构风格	隐式调用架构风格
机器学习流程定义灵活性	可通过灵活的自定义规则实现流程重组，适配“拖拽组件定义流程”的灵活需求。	流程需按固定管道-过滤器逻辑定义，灵活性不足。	基于事件触发，流程定义的灵活性受事件耦合关系限制。
学习算法可扩展性	包含解释引擎、代码存储区等组件，可通过新增规则实现算法扩展。	过滤器组件相对独立，可扩展但需遵循管道数据格式约束。	事件触发式扩展，需关注事件间的数据交换和语义依赖。

综上，解释器架构风格在**流程定义灵活性**和**算法可扩展性**上更适配平台“拖拽组件定义流程、开发新算法组件”的核心需求，因此更适合该平台。

案例-软件架构-数据相关

高频考点

数据流图（DFD）

DFD基本图形元素

元素类型	定义与说明	图形特征
数据流	表示数据的流向，必须与加工相关；除流向/流出数据存储的数据流外，其余需明确命名	*用带箭头的线段表示*
加工	描述输入数据流到输出数据流的变换，至少有一个输入和一个输出（避免“黑洞”“奇迹”“灰洞”）	*用圆角矩形或圆形表示*
数据存储	用于存储数据，可通过文件系统或数据库系统实现	*右开口矩形或者两条平行线*
外部实体	系统外的人员、组织或系统，是数据的发源地或归宿地（如考务系统中的考生、考试中心）	*用矩形表示*

DFD设计原则

1. **数据平衡原则**
    - 父图与子图平衡：子图的输入/输出数据流需与父图对应加工的输入/输出数据流一致。
    - 图内平衡：每个加工需同时具备输入和输出数据流。
2. **数据流相关原则**
    - 数据流流向包括：加工间流转、加工与数据存储间读写、外部实体与加工间输入输出。
    - 限制：外部实体间、外部实体与数据存储间、数据存储间不存在数据流。
3. **加工相关原则**
    - 每个加工必须有输入和输出数据流，且输入、输出数据流名称不能相同。

DFD元素补充方法

元素类型	补充方法
外部实体	结合与系统的交互事件、数据流，从上下文DFD中推导（如人员、组织、外部系统）
数据存储	从0层DFD中数据存储的输入/输出数据流判断，通常命名为“输入数据流名+表/信息表/文件”
加工	与信息系统功能标题一一对应，需结合数据平衡原则分析

数据字典（DD）

为数据流图中的数据流、文件、加工及数据项提供说明，通过特定符号定义数据结构。

- **符号说明（表格）**：

符号	含义	举例说明
=	被定义为	-
+	与	x=a+b，表示x由a和b组成
[.....]或[...	...]	或
{...}	重复	x={a}，表示x由0个或多个a组成
(...)	可选	x=(a)，表示a可在x中出现或不出现

- **示例**：机票=姓名+日期+航班号+起点+终点+费用；终点=[上海|深圳|北京]

二、E-R图（实体-联系图）

- **实体**：用矩形表示，由属性（候选键、主键、外键）描述，实体集是相同属性的实体集合。
  - 候选键：唯一标识元组的属性集。
  - 主键：从候选键中选一个作为唯一标识。
  - 外键：在另一关系模式中作为主键的属性。
- **联系**：用菱形表示，实体集间的对应关系，分三种类型：
  - 一对一（1:1）：如学校与校长，一个学校只有一名校长，一名校长只在一个学校工作。
  - 一对多（1:n或1:*）：如学校与学生，一个学校有多个学生，一个学生只在一个学校上课。
  - 多对多（m:n或:）：如学生与课程，一名学生可选多门课程，一门课程可被多名学生选修，这种联系会生成新的关系模式，包含双方主键和自身特有属性。

E-R图构件（表格）

构件	说明
矩形	表示实体集
双边矩形	表示弱实体集
菱形	表示联系集
双边菱形	表示弱实体集对应的标识性联系
椭圆	表示属性
线段	连接属性与实体集，或实体集与联系集
双椭圆	表示多值属性
虚椭圆	表示派生属性
双线	表示实体全部参与到联系集中

UML 四种关系（表格）

名称	子集	解释	举例	图形
关联	关联	类之间语义联系，描述对象间连接	人和公司的关联	带多重度（如0..1、0..*）的线段
	聚合	整体-部分关系，部分可独立存在	狼与狼群	带空心菱形的线段
	组合	整体-部分关系，部分不可独立存在	车轮与车	带实心菱形的线段
泛化	-	父类与子类的特殊-一般关系	人与老师的继承	带空心箭头的线段
实现	-	接口与实现类/组件的关系	-	带虚线空心箭头的线段
依赖	-	一个事物语义依赖于另一个事物的变化	人依赖食物	带虚线箭头的线段

1. 图的分类

- **结构图（静态）**：类图、对象图、构件图、部署图、组合结构图、包图。
- **行为图（动态）**：用例图、活动图、状态图、交互图（含顺序图、通信图、交互概览图、时序图）。

2. 重点图说明

- **用例图**：描述用例、参与者及关系，用于系统语境和需求建模。用例关系有包含（<>）、扩展（<>）、泛化。例如学生登记系统中，入学登记包含研讨会登记，学生登记扩展安全检查，登记学生与登记家庭成员存在泛化关系。
- **类图**：描述类、特性及类间关系，包含类的属性、操作，以及关联、聚合、泛化、依赖、接口等元素。例如公司-部门-人员的类图中，体现了聚合、关联、泛化、依赖和接口关系。
- **顺序图**：又称序列图，描述对象间消息发送与接收的顺序，包含同步消息（实心三角箭头）、异步消息（空心箭头）、返回消息（虚线箭头）。例如数据库代理的顺序图，展示了对象间消息交互的时间轨迹。
- **活动图**：描述过程和并行行为，是特殊的状态图，强调活动间的控制流程，包含并发分岔、并发汇合、监护表达式、分支等元素。例如订单处理的活动图，展示了收到订单后并行执行填写订单和发送费用清单的流程。
- **状态图**：描述对象状态及转换，由状态、转换、事件组成，关注对象行为的事件顺序。例如接电话的状态图，包含空闲、传输等状态，以及响铃、校验和等事件触发的转换。

软件项目进度安排的两种图形方法

方法	甘特图（Gantt图）	项目计划评审技术图（PERT图）
形式	水平条形图，横坐标为时间，纵坐标为任务，水平线段表示任务的开始、结束时间及持续时长	活动图，顶点表示里程碑，边表示活动，边上权重为活动持续时间（天）
核心作用	清晰描述每个任务的时间安排、进展情况及任务并行性	描述任务间的依赖关系，确定关键路径和松弛时间
关键概念	- 优点：直观展示任务时间线和并行性 - 缺点：无法清晰反映任务依赖关系，难以确定项目关键路径	- 关键路径：从开始到结束的最长路径，决定项目最短工期 - 松弛时间：最迟开始时间 - 最早开始时间，关键路径上活动的松弛时间为0
应用场景	用于项目进度的直观跟踪，让团队快速了解任务时间分布	用于复杂项目的任务依赖分析、关键路径识别，以优化项目工期
总结	侧重任务的时间重叠关系	侧重任务间的依赖关系

例题一

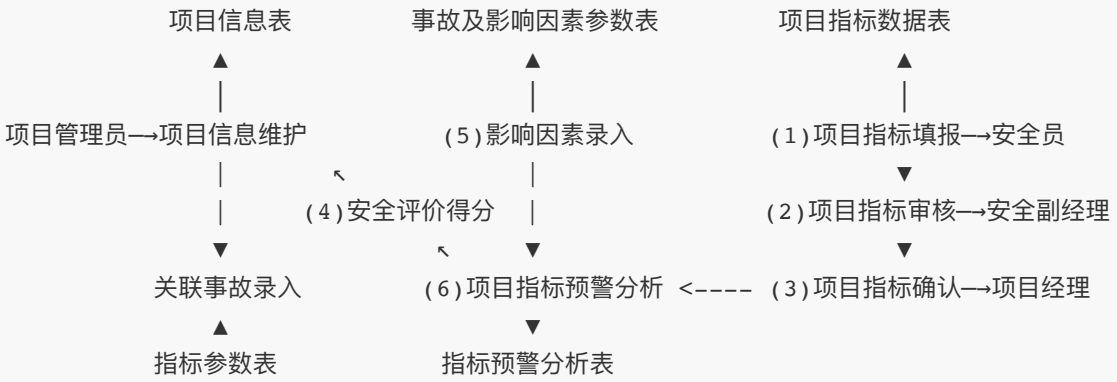
煤矿建设项目安全预警系统

【题干说明】
某能源企业拟开发煤矿建设项目安全预警系统，核心功能包括：
(a)项目信息维护
(b)影响因素录入
(c)关联事故录入
(d)安全评价得分
(e)项目指标预警分析
(f)项目指标填报
(g)项目指标审核
(h)项目指标确认

【问题1】（9分）

补充数据流图(1)~(6)的内容，并说明DFD分层细化的**数据平衡原则**。

数据流图



数据平衡原则：

- **层间平衡**：子图边界的输入/输出数据流，需与父图对应加工的输入/输出数据流一致。
- **图内平衡**：每个加工必须同时有输入和输出数据流，避免“黑洞（有输入无输出）”“奇迹（无输入有输出）”“灰洞（输入不足输出）”。

【问题2】
补充E-R图实体(1)~(6)的内容。

参照前面的数据流图填写即可

【问题3】
说明数据流图和数据字典在需求分析、设计阶段的作用。

- **数据流图**：
  需求分析阶段：建立系统功能模型，完成需求分析；
  设计阶段：为模块划分、模块接口设计提供依据。
- **数据字典**：
  需求分析阶段：对数据流图中的数据流、文件、加工、数据项做说明；
  设计阶段：依据数据存储描述完成数据库设计。

例题二

【题干说明】
某医院开发预约挂号管理系统，核心功能包括：
(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
(j)信用管理

【问题1】（6分）
补充用例图的参与者与用例名称。

根据题干描述分析用例角色，通常是某类人；然后划分用例，根据个数、共享关系确定填写位置

问题1答案：
(1) **系统操作员（系统管理员）**
(2) **预约人（患者）**
(3) **(a)注册登录**
(4)-(8) **(c)账号管理、(f)号源管理、(h)预约管理、(i)报表管理、(j)信用管理（顺序无关）**
(9)-(12) **(b)信息浏览、(d)预约挂号、(e)查询与取消预约、(g)报告查询（顺序无关）**

【问题2】
补充顺序图的对象与消息名称，并说明协作图与顺序图的区别。

顺序图信息在题干

问题2答案：
(1) **预约人员（患者）**
(2) **发起预约挂号请求**
(3) **显示医生出诊时段**
(4) **显示是否预约成功**

协作图与顺序图的区别：

- 顺序图：强调对象间交互的**消息时间顺序**。
- 协作图：强调接收/发送消息的**对象结构组织**，侧重通信方式。

【问题3】
介绍面向对象开发的3种模型（对象、动态、功能），说明其关联及需求分析中的适用情况。

1. **模型定义**
    - **对象模型**：描述对象的静态结构（关系、属性、操作），用**对象图**实现。
    - **动态模型**：描述与时间/操作顺序相关的系统特征（事件、状态等），用**状态图**实现。
    - **功能模型**：描述输入到输出的计算逻辑（不考虑顺序），用**数据流图（DFD）**实现。
2. **关联关系**
   功能模型描述“发生了什么”，动态模型确定“什么时候发生”，对象模型确定“发生的客体”。
3. **需求分析适用性**
   上述**对象模型、动态模型、功能模型均可用于需求分析**。

案例-数据库系统设计

高频知识

ORM技术

ORM（对象关系映射）是关系型数据库与对象的映射技术，开发者可通过操作对象替代复杂SQL语句，实现面向对象编程与数据库操作的统一。映射关系为：

- 数据库表 → 类
- 表记录 → 对象
- 表字段 → 对象属性

维度	优点	缺点
开发成本	降低学习与开发成本	难处理复杂查询
代码效率	无需编写SQL，减少代码量	性能低于直接使用SQL
代码质量	降低SQL质量差的影响	-

数据库类型比较

- **关系型数据库**：基于关系模型，以二维表格存储数据，支持SQL查询（如MySQL、Oracle）。
- **NoSQL数据库**：非关系型数据库，适配大数据、高并发场景，包括键值型（Redis）、文档型（MongoDB）等。
- **内存数据库**：数据存储于内存，性能极高（如Redis）。

特征	关系型数据库	NoSQL数据库
并发支持	支持并发但效率低	并发性能高
存储与查询	关系表+SQL查询	海量数据存储+查询效率高
扩展方式	向上扩展（升级硬件）	向外扩展（增加节点）
数据一致性	实时一致性	弱一致性
数据类型	结构化数据	非结构化数据
事务性	高事务性	弱事务性
数据容量	有限数据	海量数据

缓存技术

Redis：开源的内存型Key-Value数据库，支持持久化，其数据类型及场景如下：

数据类型	说明	应用场景
string	字符串/整数/浮点数	缓存、计数器、分布式锁
list	字符串列表（模拟栈/队列）	评论、点赞列表
set	无序不重复集合	用户标签、抽奖程序
hash	键值对集合	用户信息、购物车
zset	带分数的有序集合	热门帖子排名

新版本新增类型：bitmap（位操作）、hyperloglog（概率统计）、geo（地理位置）、stream（消息队列）。

MemCache：高性能分布式内存缓存系统，通过内存Hash表存储数据（如图片、数据库结果），用于减轻数据库负载。

特征	Redis	MemCache
数据类型	丰富的数据结构（String/List/Set等）	简单key/value结构
持久性	支持（RDB/AOF）	不支持
分布式存储	多种方式（主从、Sentinel、Cluster）	客户端哈希分片/一致性哈希分片
多线程支持	不支持	支持
内存管理	无（内存不足时可换至磁盘）	私有内存池/内存池
事务支持	有限支持	不支持

缓存中的常见问题及解决方法：

问题	定义	解决方式
缓存击穿	高并发下，热点数据缓存失效，大量请求涌入数据库	互斥锁、分布式锁、热点数据预加载
缓存雪崩	缓存层整体失效，大量请求直接访问后端	设置不同过期时间、使用多个独立缓存集群
缓存穿透	请求查询不存在于缓存和数据库的键，导致请求直达数据库	布隆过滤器（判断键是否有效）

分布式锁

分布式锁是分布式系统中用于控制多个节点共享资源访问的同步机制，目的是避免多节点同时操作数据导致的不一致性。

实现方式	核心原理
MySQL	基于唯一索引（利用索引的唯一性实现互斥）
ZooKeeper	基于临时有序节点（节点创建的唯一性+有序性实现锁）
Redis	基于`setnx`命令（key不存在时加锁成功）

Redis实现分布式锁流程

- **加锁**：用`setnx`命令加锁，同时通过`expire`设置超时时间（避免死锁），锁的value为随机UUID（用于标识锁的归属）。
- **释放锁**：通过UUID验证锁归属，确认后执行`delete`释放锁。

死锁解决策略：设置超时时间、使用Redlock算法、锁粒度拆分、一致性哈希等。

数据库不规范化的四大问题

以关系模式`R(sname,cname,tname,taddress)`（学生姓名、课程名、教师姓名、教师地址）为例，不规范化会导致：

问题	表现
数据冗余	同一课程的教师信息随选课学生重复存储（如100个学生选课时，教师信息重复100次）
修改异常	修改教师地址需更新所有相关记录，否则出现数据不一致
插入异常	若未知选课学生，教师的授课信息无法插入数据库
删除异常	删除学生选课信息时，可能误删课程、教师等关联数据

反规范化技术

反规范化是牺牲部分规范化以提升数据库性能的技术，核心是通过冗余减少查询开销。

维度	优点	缺点
性能	减少连接操作、降低索引/外键数量，提升查询效率	数据重复存储，浪费磁盘空间
维护	-	增加数据维护复杂度，降低修改速度

具体方法

方法	说明	作用
增加冗余列	在多表中保留相同列	减少查询时的连接操作
增加派生列	增加由其他数据计算生成的列	避免查询时的计算/聚合操作
重新组表	将多表的常用连接结果合并为新表	减少查询时的连接操作
水平分割表	按数据值将表拆分为多个独立表	适配大表、数据分散存储场景
垂直分割表	按列拆分表（主键+部分列拆分）	减少查询时的I/O次数

例题一

【题干说明】
某电商B2B系统因跨仓储调货延迟，需建设全国仓储货物管理系统：

- 核心功能：仓储常规管理、订单数据分析、货物配置预测。
- 性能要求：用户订单的货物起运地/送达时间反馈响应<1秒。
- 技术方案：缓存集群存储仓储基础信息、商品类别、库存数量；数据库存储其他商品信息。

【问题1】缓存与数据库的数据一致性方案

方案	思路
实时同步更新	同一事务内完成“删除缓存→更新数据库→写入缓存”
异步准实时更新	数据库更新时记录日志，异步排队更新缓存

选型结论：应采用**异步准实时更新方案**，原因：项目数据量大、性能要求高，同步方案并发时性能不可控，异步方案更适配高并发场景。

【问题2】缓存分片算法

算法	原理	特点
哈希分片算法	对Key做哈希计算，再对缓存节点数取余，确定存储节点	简单易理解；节点增减时缓存大面积失效
一致性哈希分片算法	将哈希空间映射为虚拟圆环，节点与Key分别哈希后，Key沿圆环顺时针匹配首个节点	节点增减时仅少量数据迁移；扩展性强

选型原因：采用**一致性哈希算法**，因其可解决节点增减时的缓存命中率下降问题，适配业务中仓储/商品数量增长的场景。

【问题3】布隆过滤器

布隆过滤器是概率型数据结构：

- 插入：通过K个哈希函数将元素映射为位数组的K个点，置为1。
- 查询：检查对应K个点是否全为1（全1则“可能存在”，有0则“一定不存在”）。

维度	优点	缺点
资源与性能	内存占用小；增查时间复杂度O(K)，速度快	存在假阳性误判；无法直接获取元素
功能特性	支持交/并/差运算；适配超大数据量	一般无法删除元素；计数删除可能出现回绕

例题二

【题干说明】
某医药企业建设线上药品销售系统，功能包括药品展示、订单、用户互动、热销排名等，采用MySQL+Redis混合架构。规范设计后的表结构：

- 供应商（供应商ID，供应商名称，联系方式，供应商地址）
- 药品（药品ID，药品名称，型号，价格，供应商ID）
- 药品库存（药品ID，当前库存数量）
- 订单（订单号码，药品ID，供应商ID，数量，金额）

【问题1】反规范化设计

方法	说明
增加冗余列	在多表中保留相同列，减少查询连接
增加派生列	增加计算生成的列，避免查询时计算
重新组表	合并多表常用连接结果为新表，减少连接
水平分割表	按数据值拆分大表为独立表
垂直分割表	按列拆分表，减少查询I/O

选型结论：用户查询商品信息应采用**增加冗余列**方法（将供应商名称、库存数量冗余到药品表），减少多表关联查询。

【问题2】反规范化的数据一致性解决方法

方法	说明	适用场景
批处理维护	定期运行作业/存储过程更新数据	实时性要求低
应用逻辑	同一事务中操作所有涉及表	逻辑易遗漏，维护复杂
触发器	数据修改时立即触发对应列更新	实时性好、易维护

选型结论：该系统应采用**触发器**，适配药品销售的实时数据一致性需求。

【问题3】Redis相关设计

热销药品排名的数据类型：选择**ZSet（有序集合）**，因其支持按分数排序，适配“按销量排名”的场景。

Redis与MySQL数据同步方案：

方案	说明
事务同步	利用MySQL事务一致性，事务提交后更新缓存
访问控制位	数据库变化时更新控制位，查询时先判断控制位，有变化则查库更新缓存
中间件同步	通过数据库中间件（如Canal）监听binlog，实时同步缓存