操作系统

操作系统的用户接口

操作系统为用户提供 3 个接口：

（1）命令接口：提供用户一组命令用来直接或间接地组织和控制作业的执行或者管理计算机系统（直接-联机方式|间接-脱机方式）

（2）程序接口：由一组系统调用命令组成，提供一组系统调用命令供用户程序使用，编程人员可以使用它们来请求操作系统服务。

（3）图形接口：通过图标、窗口、菜单、对话框及其他元素和文字组合，在桌面上形成一个直观易懂、使用方便的计算机操作环境。

进程死锁的必要条件

必要条件有三

互斥：即当某个程序进入临界区占用公共资源时，其它程序不能进入该临界区使用公共资源，需等到该进程完成操作退出临界区后才可使用；

不抢占：即当有进程进入临界区域占用公共资源时，其他进程不能采取抢占的方式；

等待且占有：当一个进程在等待其他进程时，继续占有已经分配的资源

再加一条的话就是循环等待：存在一个进程链，使得一个进程占有下一个进程所需要的资源。

操作系统引入进程的作用和意义

从理论角度来讲，进程是一个动态的有生命周期的程序的抽象；

从现实角度看，进程是一种数据结构，引入进程可以更清晰地刻画动态系统的内在规律，可以更加有效地管理和调度进入操作系统的程序。

从宏观的角度看，引入进程就是为了实现并发。

进程的状态

基本状态：运行态、就绪态、阻塞态

由于创建和完成又可以加上新建态和推出态

由于交换和虚拟内存技术，可以将就绪态和阻塞态细化为：就绪态和挂起就绪态、阻塞态和挂起阻塞态。

并发进程要解决的问题

并发进程需要解决的问题主要是互斥和同步，其次是死锁和饥饿

互斥：当一个进程进入临界区使用公共资源时，其他进程不能进入临界区使用该资源

同步：在某些特定情况下，某些进程必须严格按照特定顺序执行

死锁：两个进程相互持有彼此需要的资源，而导致进程不能继续执行的现象

饥饿：在等待的进程由于优先级等问题长时间不能从等待队列进入运行态的现象

操作系统的主要特性

操作系统是使可以使用户方便有效地管理计算机系统，并能够为应用程序提供统一化平台的一系列程序集合，具有易扩展性。

四个主要特性：

并发：可以实习多个线程同时进行

共享：某些资源可以被多个线程或者用户共享

虚拟：通过某种技术将一个物理实体变为多个逻辑体

异步：在单处理器多通道程序设计或多处理器中，允许程序不是从一而终地执行下来，可以有交叉或者走走停停

操作系统的作用：

（1）控制管理计算机的所有硬件软件资源，方便用户使用
（2）合理组织计算机内部各部件协调工作、各资源协调分配
（3）为使用者提供统一的编辑、操作平台，同时为应用程序提供统一平台

解释进程通信，并列出四种进程间的通信方式

进程间的交互程度：
相互完全独立
间接联系:不知道彼此进程 ID，通过共享一部分资源知道彼此存在，比如 I/O 设备
直接联系：进程间通过进程 ID 相互通信，合作完成某些活动
进程间通信方式：
（1）管道/管程：一种半双工的通信方式，通常用于具有父子关系的进程之间，数据只能单向流动；
（2）信号量：信号量是进程间相互通信的一个整数计算器，用来控制多个进程对共享资源的访问。信号量上定义了三种操作方式：初始化、递增和递减。递增用于释放一个阻塞进程，递减用于阻塞一个进程。
（3）消息队列：消息队列是由消息构成的链表，存放在内核中，并由消息队列标识符标识。消息队列克服了信号量传递信号少等缺点，且可在分布式操作系统和多处理器系统中实现。
（4）共享内存：即建立一段给其他进程访问的内存，该内存由一个进程建立，但是多个进程可以访问。共享内存的最快的 IPC 方式。
（5）信号：比较复杂的通信方式，用于通知接收进程某个时间已经发生。

进程死锁的原因、条件、解决方法

进程死锁是指多个进程相互持有彼此需要的资源，不可抢占，从而无法继续执行程序的现象。
条件有四：互斥，等待且占有、不可抢占、循环等待
死锁检测:通过死锁检测算法检测循环等待的发生，再试图处理死锁
死锁预防：针对四个条件进行修正。互斥（不可改变）；占有且等待（低效方法，差异性分配所有资源）；不可抢占（允许抢占或占了的先吐出来）；循环等待（定义资源类型的先行顺序来预防）
死锁避免：导致死锁的进程不启动、不分配

中断在操作系统中的作用

中断是指在计算机运行期间，系统内部发生某些急需处理的时间，使得处理器暂时停止当前正在执行的程序，转而去执行当前紧急的事件处理程序，并在处理完后返回原理被中断处，继续运行的过程。

中断可以暂时停止一个处理器任务，提高处理器的利用率和吞吐量，使得多道程序实际成为可能。

解释操作系统的两种用户接口

减少页表占有内存空间的方法

倒排页表，两级页表，多级页表

文件系统的作用

文件系统是操作系统的一个重要部分，用来明确磁盘和分区上的文件和数据结构，即在磁盘上组织文件的方法或程序。

文件系统由三个部分构成：管理软件、被管理软件、想要数据结构。其作用是为用户建立文件，控制和管理文件：实现逻辑文件和物理文件之间的转换；有效地分配文件的存储空

间；实现对文件地操作管理；实现文件信息的共享和保护；提供用户接口

叙述处理器调度的主要目标和原则，描述三种不同目标的调度算法及其调度依据。

在多道程序设计中，为了支持并发和异步，主存中有多个进程。这就要求操作系统能够按照某种算法动态地把处理器分配给就绪态队列中的进程，使其运行。这就要求处理器的调度能够保证处理器的利用率以及改善系统性能。

准则：周转速度快、系统吞吐量高、各类资源平衡等

处理器调度的级别分为三级：高级、中级、低级或长程、中程、短程。长程调度决定哪些进程/作业可以进入系统中处理，它控制系统并发度；中程调度即交换或者虚拟中的页面调度；短程调度最频繁，直接精确决定下次要执行哪个进程。

调度算法：先来先服务；短作业优先；高优先权优先；时间片轮转

程序员编程可利用操作系统的什么接口，可用哪些功能。

面向程序员的接口是程序接口，可进行系统调用

设备驱动程序的作用和实现方法

设备驱动程序是一个小型的系统级程序，它能够使特定的硬件和软件与操作系统建立联系，让操作系统能够正常运行并启动该设备。有了设备驱动程序吗，操作系统可以明确知道设备是什么设备，有什么功能。

操作系统内核的含义、作用和分类

内核是基于硬件的第一次软件扩充，提供操作系统的最基本功能，是操作系统工作的基础，负责管理系统的进程、内存、设备驱动程序、文件和网络系统，决定着系统的性能和稳定性。

分类：单内核、微内核、混合内核、外内核

说明信号量和 PV 操作的含义和用法

PV 操作指的是通过释放。即解释信号量在进程通信中的使用。

页面替换算法，按照缺页中断率由低到高

最佳置换，久未使用则退，先进先出

解释文件的静态共享和动态共享

静态共享是文件的可执行程序和文件打包一起组成一个体积大的文件分享。使用该文件时就会把所有的模块加噪，用到时很快就执行，效率高。

动态共享是指扩展模块作为一个独立的存在，当使用该模块时再调用它。核心文件比较少，模块随时用随时加噪，耗费内存相对较少。

从支持虚拟储存管理的角度给出 MMU 的三个以上主要功能

MMU，Memory Management Unit,内存管理单元，是中央处理器用来管理虚拟存储器、物理存储器的控制线路，同时负责虚拟地址映射为物理地址，以及提供硬件机制的内存访问授权，多用户多进程操作系统。

对内存管理有五点需求：重定位、保护、共享、逻辑组织和物理组织

四个主要功能：内存分配、内存保护、内存扩充、地址映射（将虚拟地址映射为物理地址，实现虚拟内存；能给不同的地址空间设置不同的访问属性）

给出 I/O 软件系统的层次

用户层、设备独立性软件、设备驱动程序、中断处理程序、硬件

数据库

数据库系统=相互关联的数据的集合（数据库）+访问这些数据的程序
数据库的主要内容：
数据视图
数据模型：用于描述数据、数据关系等的工具的集合，通常分为 4 类
1.关系模型：工具-表用于记录；关系模型-表的集合
2.E-R 模型：实体-联系模型基于对现实世界的一种认识，提供一个找出在数据库中表示实体以及实体间如何关联的方法。
3. 对象-关系模型：E-R 模型的扩展版
4.半结构化数据类型：允许相同类型的数据含有不同的数据定义
数据库语言：数据库定义语言（DDL）：定义数据库模型
数据库操作语言（DML）查询，更新（添加、删除、修改）

数据库系统设计的目的：为用户提供高效方便使用信息的方法；为用户提供数据关系的抽象视图；保证数据的独立性
数据独立性=物理独立性+逻辑独立性
物理独立性：应用程序与存储器中的数据保持独立关系
逻辑独立性：应用程序与数据库中的逻辑结构保持独立关系
数据和程序的独立，简化了程序，减少了应用程序的修改和维护。
数据库管理系统通过**三级模式结构和两级映射**实现数据的独立性。三级模式结构包括外模式、概念模式和内模式。两级映射是指当整个系统要求改变模式时（增加记录类型、增加数据项），由 DBMS 对各个外模式/概念模式的映像做相应改变，从而保证了数据的逻辑独立性；当数据的存储结构改变时，由 DBMS 对概念模式/内模式的映像做相应改变，从而保证了数据的物理独立性。
文件处理系统：DNS 诞生之前的“数据库系统”，是数据和程序的简单堆积。
缺点：数据冗余和不一致；数据访问困难；数据孤立；完整性问题（数据库中的值必须满足某些一致性约束）；原子性问题（事件/指令发生故障时可以恢复）；并发访问异常；安全性问题。
事务：数据库中一些操作的集合称为单元，单一逻辑工作单元的操作集合称作事务。
事务具有以下性质：
原子性：一个事务的执行只有执行成功和不执行
隔离性：并发时，一个事务的程序不会被其他事务干扰
一致性：数据库中的数据在执行一个事务前后都必须保持一致
持久性：一个事务完成后，它对数据库的改变是永久的
事务的隔离性即操作系统中的互斥，为并发操作提供最基本的保证，具有提高吞吐量和资源利用率、减少等待时间的优点。并发通过并发控制机制来实现程序并发，其目的是通过调度实现事务的可串行化。
事务的隔离性级别有如下几个：可串行化；可重复读（只允许读已提交的数据但可以重复）；已提交读（只允许读已提交的数据但不可以重复）；未提交读（允许读未提交的数据，最低的一致性级别）
实现并发且不允许脏写的方法：锁、时间戳、快照隔离
恢复系统可保证事务的原子性和持久性，主要做到事务程序发生故障时，**数据不丢失且快速**

恢复。

形式化关系查询语言

过程化语言：关系代数: 一元运算：更名、投影、选择
二元运算：并、笛卡尔积、差
附加运算：交、赋值、自然链接、聚合

非过程化语言：元组关系、域关系

大数据：指无法在一定时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能处理，具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产。

适应于大数据的技术：大规模并行处理数据库，数据挖掘电网，分布式文件系统，分布式数据库，云计算平台，互联网，可扩展的存储系统。

大数据的 5V 特点：Volume(大量)、Velocity(高速)、Variety(多样)、Value(低价值密度)、Veracity(真实性)。大数据采用所有数据进行分析处理，不用随机分析法（抽样调查）。

大数据的作用：

（1）大数据的处理分析正在成为新一代信息技术融合应用的结点。移动互联网、物联网、数字家庭、电子商务等是新一代信息技术的应用形态，这些应用不断产生大数据。云计算为这些海量、多样化的大数据提供存储和运算平台。通过对不同来源数据的管理、处理、分析与优化，将结果反馈到上述应用中，将创作出巨大的经济和社会价值。大数据具有催生社会变革的能量，但释放这种能量，需要严谨的数据治理、富有洞见的数据分析和激发管理创新的环境。

（2）大数据是信息产业持续高速增长的新引擎

（3）大数据利用将成为提高核心竞争力的关键因素

（4）大数据时代科学研究的方法手段将发生重大改变。大数据十大可通过实时监测、跟踪研究对象在互联网上产生的海量行为数据，进行挖掘分析，揭示出规律性的东西，提出研究结论和对策。

大数据时代所面临的问题：容量；延迟；安全；成本；灵活性；数据的积累和一致性问题。

数据库管理系统(DBMS)的主要功能：

（1）数据定义功能：DBMS 提供相应数据语言（DDL）定义数据库结构，刻画数据库框架，并被保存在数据字典中

（2）数据存取功能：DBMS 提供数据操纵语言（DML），实现对数据库数据的基本存取操作：检索、插入、修改和删除

（3）数据库运行管理功能：DBMS 提供数据控制功能，即是数据的安全性、完整性和并发控制等对数据库运行有效的控制和管理，以确保数据正确有效

（4）数据库的建立和维护功能：包括数据库初始库的装入，数据库的转储、恢复、重组织，系统性能监视、分析等功能

（5）数据库的传输：DBMS 提供数据的传输，实现用户程序与 DBMS 之间的通信，通常与操作系统协调完成。

NoSQL 数据库是一种与关系数据库截然不同的数据库管理系统，它的数据存储格式可以是松散的、通常不支持 Join 操作，并且支持横向扩展，也称为非关系数据库。

云计算时代或大数据时代对数据库技术提出了新的需求，主要体现在：海量级数据；大规模集群管理，分布式应用可以更加简单地部署、应用和管理；低延迟读写速度；建设及运营成本。

传统关系型数据库的缺点：支持容量有限；高并发读写速度慢；扩展性差；建设和运营成本高。

NoSQL 数据库是一种应大数据时代而产生的数据库，非常关注对数据高并发读写和海量数据的存储，在架构和数据模型方面做了简化，在扩展和并发方面做了增强。
其常用模型有以下三种：
列式数据库（Column-Oriented），存储数据时围绕着列
键-值(Key-Value),一个 key 对应一个 value
文档（Document）
数据库的完整性是指数据的正确性、一致性和相容性，确保数据库中的数据可以成功和正确地更新，防止错误的数据进入数据库造成无效操作。
数据完整性包括：实体完整性、域完整性、参照完整性和用户定义完整性。
实现数据完整性的措施主要包括：
（1）约束。定义了可输入表或者表的单个列中数据的限制条件，约束独立于表结构；
（2）默认值。在没有为某列指定数据时，使用默认指定数值；
（3）规则。规则是对已存入数据的规定和限制
（4）属性约束。包括非空值约束、域约束、检查子句等
（5）全局约束
不正确的并发导致数据不一致性，主要体现在丢失修改、读“脏”数据、不可重复读和产生“幽灵”数据：
丢失修改：两个事务一起读入并修改某一数据，后写入的修改了先写入数据；
脏数据：指不正确数据。事务 A 要读取事务 B 写入的数据，B 成功写入但未成功提交，数据恢复到 B 执行前，但数据被 A 读取，产生不正确数据；
不可重复读：前一个事务修改了数据，导致后面的事务不能再读写该数据；

软件工程

软件工程：（1）将工程化方法用于软件；（2）对（1）中的方法进行研究
软件过程：开发软件产品或软件系统时，所要执行的步骤或流程
过程模型指开发的步骤或者流程的通用模板，即为了改变软件开发的混乱状况设计的模板
过程模式包含几点：模式名称、驱动力、类型、启动条件、问题、解决方案、结果、相关模式、已知应用和实例。
UML：统一建模语言，是一种用来对真实世界物体进行建模的语言。具有良定义、易于表达、功能强大、使用普遍等特点。UML 定义了五类 10 种模型图
用例图（展示系统外部的各类执行者与系统提供者的各种用例关系）、类图（展示系统中类的静态结构）、对象图、包图、状态图（描述一类对象具有的所有可能的状态及其转移关系）、顺序图、合作图、活动图、构建图、配置图
敏捷性：适当（或快速）响应变更
软件系统结构：建立计算机系统所需的数据结构和程序构件。软件设计的目标之一就是导出系统体系结构示意图，该示意图为一个框架，将指导更详细的设计活动。
设计模式：描述了解决某个特定问题的设计结构，该设计问题具有特定条件，该条件会影响到模式的应用和使用方式。如体系结构模式、构件级设计模式、用户界面设计模式、WebAPP 设计模型，移动 APP 设计模式
关注点分离：关注点是一个特征或者行为被指定为软件需求模型的一部分。将关注点分离为更小的关注点即关注点分离（分而治之+更容易解决问题）
模块化：模块化是关注点分离最常见的表现
信息屏蔽：每个模块对其他设计模块都屏蔽自己的设计决策。信息屏蔽是描述模块化独立性

的概念，即模块应该被特殊说明并设计，使信息（算法和数据）都包含在模块内，其他模块无需对这些信息进行访问。

重构：简化构件的设计（代码），而无需改变其功能或者行为。

设计类：分析模型定义了一组分析类，每个分析类都描述问题域中的某些元素，这些元素关注用户可见的问题方面。五种不同的设计类：用户接口类、业务域类、过程类、持久类、系统类。

面向对象设计是一种软件设计方法，它解决类与相互通信的对象之间的组织关系，它是独立于编程语言的，但是面向对象设计模式的最终实现仍要使用面向对象编程语言来表达，如VB，.Net,C++等。其本质是以建立模型体现出来的抽象思维过程和面向对象的方法。模型是用来反映现实世界中事务特征的。任何一个模型都不可能反映客观事物的一切具体特征，只能对事物特征和变化规律的一种抽象，且在它所涉及的范围内更普遍、更集中、更加深刻地描述客体的特征。通过建立模型而达到的抽象是人们对客体认识的深化。

面向对象设计的三个主要特征：

（1）封装性：即信息屏蔽

（2）继承性：继承是一种由已有类创建子类的机制，利用集成可以先创建一个共有属性的一般类，根据整个类再创建具有特殊属性的子类，被继承的类称为子类，当然子类也可以成为父类继续向下扩展。

（3）多态性：同一个信息被不同的对象接收到时可能产生不同的行为

面向对象设计（设计基于类的构件）五个原则：

单一职责原则：一个类只有一种单一的功能

开放封闭原则：软件对扩展开放，对修改封闭

Liskov 替换原则：子类必须能够替换其父类，但是父类不一定能替换子类

依赖倒置原则：依赖于抽象：高层模块-抽象-底层模块

接口隔离原则：使用多个小的专用接口，而不是一个大的总接口，即一个类对另一个类的依赖应该建立再最小接口上。

面向对象的设计模式：创建型模式、结构型模式、行为型模型

软件系统结构的重要性：提供了一种表示，方便以后的交流学习；突出了早期的设计决策；构建了一个相对小、易于理解的模型，该模型描述了系统如何构成、构建如何一起工作。

体系结构模式：描述了解决某个特定设计问题的设计结构，该设计问题具有特定条件，该条件会影响到模式的应用和使用方式，即设计模式。

体系结构风格：不同于结构模式，为整个系统的所有构件建立一个结构。模式涉及的范围要小一些,更多集中在体系结构的某一方面而不是体系结构整体;模式在体系结构上施加规则，描述更面向具体问题。

体系结构风格分类：以数据为中心的体系结构；数据流体系结构；调用和返回体系结构；面向对象体系结构；层次体系结构

构件：系统中模块化的、可复用的、可部署的部件封装实现并对外提供一组接口，面向软件体系架构的可复用软件模块。

框架：框架是可以重复使用的“微型体系结构”，可作为应用其他设计模式的基础。

设计模式与框架的区别：设计模式比框架更加抽象；框架中含有设计模式，设计模式中共不含有框架。联系：共同搭配完成一个完整的设计。

软件架构模式：分层模式、客户端-服务器模式、主从设备模式、管道-过滤器模式、代理模式、点对点模式、事件总线模式、模型-视图-控制器模式、黑板模式、解释器模式。

分层模式：每个模块必须属于某个层次，每个层次为下一个层次提供更高层次的服务，同时派任务给上一个层次的模块，层次间不可逆向调用、不可跨层调用。

一般信息系统中最常见的 4 层：表示层、应用层、业务逻辑层、数据访问层
使用场景：桌面应用程序、电子商务 Web 应用程序
优缺点：适应集成不同类型功能、不适应简单的业务模型
基于模式的软件设计：基于模式的软件设计也是一种软件设计方法，区别于面向对象的设计方法或传统的软件设计方法，给予了一种新的解决问题的思考方式。分为：理解全局、在抽象层上表示模型、在特定平台上进行关联。
面向对象的设计方法和基于模式的设计方法的联系区别：面向对象设计在技术上很成熟，主要特点是交互性强、具有安全的存取模式、网络通信量低、响应速度快、利于处理大量数据。该结构的程序是针对性开发，变更不够灵活，维护和管理的难度较大，分布功能弱且兼容性差，因此缺少通用性，具有较大的局限性。基于模式的设计方法主要特点是分布性强、维护方便、开发简单且共享性强、总体拥有成本低。但数据安全性问题、对服务器要求过高、数据传输速度慢、软件的个性化特点明显降低，难以实现传统模式下的特殊功能要求。此外，实现复杂的应用构造有较大的困难。
类的继承和封装之间的矛盾：封装性指的是隐藏模块化软件的信息(数据、程序、细节等)；继承性指的是子类继承父类属性并可以扩展。矛盾在于如果一个父类是封装的，子类继承父类应该依然是封装的,这与封装的不可以扩展产生矛盾。故封装性和继承性是一个平衡关系，而不是绝对关系，一定程度的继承和封装可以提高开发效率。C++使用虚拟继承的方式部分解决了继承性与封装性的矛盾。
面向对象语言实现封装：封装把信息（数据、方法）封装到一个类中，然后设置访问控制符。允许其它类访问的设置为 public；允许子类访问设置为 protected；只能自己访问的设置为 private。封装是实现抽象的手段之一。实现封装的方法有：属性的封装、类的封装和方法的封装等。
模型驱动体系架构（MDA）一种独立于特定平台和软件供应商的软件体系结构设计和开发方法，适用于设计、部署、集成等软件开发的整个生命周期。MDA 建模是基于功能而非基于特定语言、平台或实现技术，它可以简化系统集成、缩短开发周期和节省企业资源。模型通常以图和文字的形式来描述一个系统及其环境。模型驱动的方法就是利用模型来引导系统的设计、开发和维护。模型驱动架构是利用系统的模型来生成系统的体系结构。
MDA 有三个视图：
计算无关视图(CIV)：将系统基本处理逻辑同平台相关的技术规范分离开。CIV 关注于系统的环境和需求，隐藏系统的具体结构和实现。基本需求。
平台无关视图(PIV):关注系统的操作而隐藏了平台相关的细节，PIV 视图可能用一种通用的、平台无关的建模语言如 UML 来描述。逻辑层。
平台相关视图(PSV):关注特定平台的相关实现细节。实现层。

计算无关模型（CIM）通常由业务分析人员创建，展示了系统的业务模型。平台无关模型（PIM）是系统功能的模型，通常由系统架构师创建。平台相关模型（PSM）对一个或多个平台的 PIM 模型的具体实现建模。

MDA 的真正价值在于 CIM 模型可以通过简单的映射转换成 PIM 模型。同样的，PIM 模型也可以映射成 PSM 模型，而 PSM 模型则可以最终转换成具体的实现代码。PIM 和 PSM 之间是抽象与具体的关系，与具体实现技术无关，PIM 可以被多种实现技术复用，当技术平台发生变化时，PIM 不必做改动。

MDA 提供了一种优雅而可靠的开发框架，使得系统架构师在没有考虑到系统实现的任何细节之前就可以事先定义好系统的模型。
可移植性：根据已有的 PIM 创建新的 PSM 是相当容易的
跨平台操作性：可根据特殊规则将一个 PIM 模型转化为一个异构的 PSM 模型

开发效率，MDA 是一种及其高效的设计和开发方法
软件质量，使用一种单一的模型来生成和导出系统的大部分代码可以极大地降低人为
对新技术的快速包容、平台无关性、领域相关性、降低开发成本、缩短开发周期
MVC 模式主要由模型、视图和控制器三部分组成。模型保存数据和核心程序，视图用来显示数据和接收用户指令，控制器处理数据和用户指令。模型与视图无关，模型中的数据由控制器处理，然后在视图显示；视图与模型无关，视图接收的指令由控制器处理，处理的数据结果作用在模型中。实现了模型的简单化和可复用性、视图的简单化和可复用性。
订阅发布模式：事件触发者都被抽象出来，称为消息发布者，即图中的 P。事件接受者都被抽象出来，称为消息订阅者，即图中的 S。P 与 S 之间通过 S.P（订阅器）连接，实现 P 与 S 的解耦。
设计模式是一种广为人知的、可复用、经过分类编目、代码设计的经验的总结。使用设计模型是为了可重用代码，让代码更加易于理解、保证代码的可靠性，让设计者可以更加方便有效地使用代码、设计模型。
什么叫软件体系结构？

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。软件体系结构包括定义能满足所有技术和业务要求的结构化解决方案，同时优化性能、安全性和可管理性等常见的质量特性。它需要根据各种因素做出一系列决策，而每项决策都会对软件的质量、性能、可维护性和全面成功产生相当大的影响。

这一定义注重区分处理构件、数据构件和连接构件，这一方法在其他的定义和方法中基本得到保持。

软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模型以及这些模式的约束组成。软件系统结构不仅指定了系统的组织结构和拓扑结构，并且显示了系统需求和构成元素之间的对应关系，提供了一些设计决策的基本原理。

一个程序和计算机系统该软件体系结构是指系统的一个或者多个结构。结构中包括软件的构件，构件的外部可见属性以及他们之间的相互关系。体系结构并非可运行软件，而是一种表达，使软件工程师能够（1）分析设计在满足规定需求方面的有效性（2）在设计变更相对容易的阶段，考虑体系结构可能的选择方案（3）降低与软件构造相关的风险。

算法导论

简单描述 Quicksort 算法的原理与过程

Quicksort 算法是由冒泡算法改进而得到的，其基本原理是：在待排序的 n 个元素中任选一个元素作为基准，然后对数组 a[n]进行分区操作，通过一趟排序之后将数组 a[n]分为两个独立的部分，使得基准左边的元素都不大于基准值，基准右边的元素都不小于基准值，这时基准元素排在这两部分中间，调整到了排序后的正确位置。完成一趟快速排序后，采用递归方法对所得的两个子数组分别重复上述快速排序，直至每部分内只有一个元素或元素为空为止。这时数组的每个元素都被排在正确的位置，这个数组达到有序。

因此快速排序算法的核心是分区操作，即调整基准元素的位置以及调整返回基准的最终位置以便分治递归。

Quicksort 算法的过程

(1) 设置两个变量 start、end, 排序开始的时候：start=1,end=N;

(2) 以第一个数组元素作为基准元素赋值给 povit, 即 pivot=arry[1];

(3) 从 end 开始向前搜索，即由后开始向前搜索(end--), 找到第一个小于 pivot 的值 arry[end]，并与 arry[start]交换，即 swap(arry,start,end);

(4) 从 start 开始向后搜索，即由前开始向后搜索(start++), 找到第一个大于 pivot 的 arry[start]，与 arry[end]交换，即 swap(arry,start,end);

(5) 重复第 3、4 步，直到 start=end, 这时 arry[start]=arry[edn]=pivot, 而 pivot 的位置就是其在整个数组中的正确位置;

(6) 通过递归，将问题规模不断分解，将 pivot 两边分成两组继续求新的 pivot，最后解出问题。

```
public static void quick_sort_swap(int[], arry,int start, int end){
        int i = start, j = end;
        int mid = array[(start + end) / 2];
        int tmp = 0;
        if (start < end){
                while (i != j){
                        while(j>i  && array[j]>mid)
                                j--;
                        while(i<j && array[i] < mid)
                                i++;
                        if (i <= j){
                                tmp = array[j];
                                arry[j] = array[i];
                                arry[i] = array[j];
                        }
                }
                quick_sort_swap(arry, start, i-1);
                quick_sort_swap(arry,j+1, end)
        }
}
```

Quicksort 算法的时间复杂度

快速排序的时间复杂度只要耗费在化分操作上，对长度为 n 的区间进行化分，共需 n-1 次关键字比较。

最小生成树算法（Kruskal、Prim）是寻找最小生成树的子集结构 A，并利用循环不等式结构不断添加权值最小的安全边，直至生成最小生成树的一种算法。根据添加安全边的方式不同，可将最小生成树算法分为 Kruskal 算法和 Prim 算法。基本设计思路：

A=空集；

While A 不是图 G 的最小生成树

    找到权值最小且对 A 安全的边(u,v)；

    A=A∪{(u,v)}

Return A //安全指该边加入不会破坏 A 式最小生成树子集的结构

Kruskal 算法输入图 G(V,E)和权值函数 w。先将 V 中的每个顶点独立为一个集合，然后将 E 中的所有边按权值大小排序，接着按权值从小到大的顺序遍历所有边，并做以下操作：判断新加入边的两个顶点是否在同一棵树下，若否，则将该边并到目标最小生成树子集 A 中。

MST-Kruskal(G,w)

A=空集；

将 V 中的每个顶点独立为一个集合(或一个无边的树)；

将 E 中的所有边按照权值从小到大的顺序进行排序；

For 从小到大顺序遍历 E 中的每一条边(u,v)

    If 顶点 u 和 v 分属不同的树结构

        Then A=A∪{(u,v)}；

            合并顶点集{u},{v}；

    End

End

Prim 算法输入图 G(V,E)，权值函数 w 和最小生成树的根结点(即初始点)r。先构造树集合 S={r}和其补集 T=V-S，再从选取链接 S 和 T 的所有边中权值最小的边(u,v)，其中 u∈S，v∈T，接着将 v 并入集合 S 中，相应的，从 T 中去掉顶点 v，并将新边(u,v)并入目标最小生成树子集 A 中。按照该循环不变式以此类推，直到 T 为空集。

MST-Prim(G,w,r)

A=空集；

S={r}；

T=V-S;

While T 不等于空集

    (u,v)为链接 S 和 T 的所有边中权值最小的边，且 u∈S，v∈T；

    S=S∪{v}；

    T=T-{v}；

    A=A∪{(u,v)}；

体现贪心策略：（1）最优化问题具有最优子结构；（2）当前做出局部最优选择后，当前子集的最优子结构循环不变。在最小生成树问题中，最优子结构为最小生成树的子集在该子图中也是最小权值树结构，每次选取当前权值最小且安全的边(即局部最优选择)，仍然满足该循环结构。Kruskal 算法是通过遍历所有边的方式，选取当前权值最小且安全的边；Prim 算法是在当前 S 和 T 的分割边中选取当前权值最小且安全的边。

Kruskal 算法计算复杂度为：O(1)+O(V)+O(E·lgE)+O(E·lgV)=O(E·lgE)，

由于 E<V^2，则计算复杂度为：O(E·lgV)；

Prim 算法计算复杂度为：O(1)+O(V·lgV)+O(E·lgV)=O(E·lgV)；如果使用 Fibonacci 堆，Prim 算法的计算复杂度可以优化为：O(1)+O(V·lgV)+O(E·1)=O(E+V·lgV)。

显然在顶点个数|V|明显小于边个数|E|时，适合采用 Prim 算法。

数学归纳和递归：直接或者间接调用本函数语句的函数称为递归函数，其必须满足两个条件：（1）每一次调用自己，更接近于解；（2）必须有边界条件或者终止准则。
联系：都包含递推的思想，在计算机算法设计中相互交织；
区别：（1）数学归纳法强调方法，递归强调应用；
（2）数学归纳法是从初始值出发，类似于 F(n){i=1;s=1;while i==n+1:s=s*i;i=i+1;return s}
递归是有边界条件的，设计时反向调用子集，如 F(n){if n==1 rerurn 1;return n*F(n-1)}
近似算法指结果不一定是最优的，但也在可以承受的范围内，而且可以比精确求解消耗更少的资源。定义：设 D 是一个最优化问题，A 是一个算法，若把 A 用于 D 的任何一个实例 I，都能在 I 的多项式时间内得到 I 的可行解，则称算法 A 为问题 D 的一个近似算法，其中 I 表示实例 I 的规模或输入长度。
随机算法是在算法中使用了随机函数，且随机函数的返回值直接或者间接地影响了算法的执行流程或执行结果。设 I 是问题 P 的一个实例，用算法解 I 的某些时刻，随机选取 b I ，由 b 来决定算法的下一步动作。
优点：执行时间和空间，小于同一问题的已知最好的确定性算法；实现比较简单，容易理解。
随机算法分为两类：
（1）拉斯维加斯（LasVegas）算法，要么给出正确解，要么无解。无解时，需要再次调用该算法。要么给出问题的正确答案，要么得不到答案。反复求解多次，可使失效的概率任意小。
（2）蒙特卡罗（Monte Carlo）算法，总能给出解，但可能是正确解也可能是错误解。多次调用该算法可降低错误率。总能得到问题的答案，偶然产生不正确的答案。重复运行，每一次都进行随机选择，可使不正确答案的概率变得任意小。