# 第一章 企业信息化战略与实施

## 1 考情分析

### 1.1 本章概要

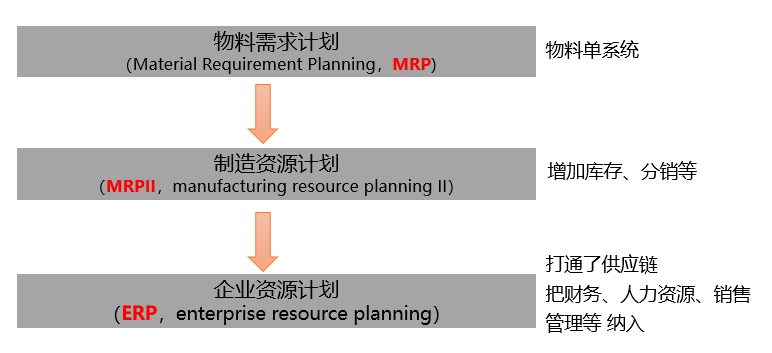
|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 企业信息化与电子商务  （⭐⭐⭐⭐） | 企业资源计划ERP |
| 2 | 客户管理CRM |
| 3 | 供应链管理SCM |
| 4 | 产品数据管理PDM |
| 5 | 商业智能BI |
| 6 | 业务流程分析方法 |
| 7 | 业务流程重组BPR |
| 8 | 业务流程管理BPM |
| 9 | 电子商务分类 |
| 1 | 企业集成（⭐⭐⭐） | 企业集成分类 |

## 2 考点精讲

### 2.1企业信息化与电子商务（⭐⭐⭐⭐）

#### 2.1.1 企业资源计划（ERP）

（1）发展过程

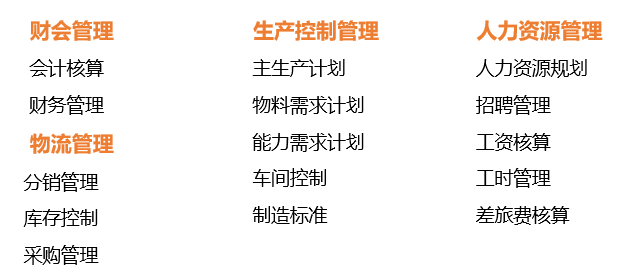


（2）ERP结构

ERP是将企业所有资源（企业三大流：物流、资金链、信息流）进行集成整合，全面一体化管理的管理信息系统。

包括三方面：生产控制（计划、制造）、物流管理（分销、采购、库存管理）和财务管理（会计核算、财务管理）。这三个系统本身就是一个集成体，它们相互之间有相应的接口，能够很好地整合在一起。

ERP系统的主要功能模块：



（3）三个理解层次

管理思想：它是管理思想的变革。

软件产品：但不是直接买来就用，需要个性化的开发与部署。

管理系统：存在众多的子系统，这些子系统有统一的规划，是互联互通的，便于事前事中监控。

#### 2.1.2 客户管理CRM

CRM理念：将客户看作资产；客户关怀是中心，目的是与客户建立长期和有效的业务关系，最大限度地增加利润；核心是客户价值管理，提高客户忠诚度和保有率。

CRM的功能

（1）客户服务：CRM的关键内容。

（2）市场营销：包括商机产生、商机获取和管理、商业活动管理和电话营销等。

（3）共享的客户资料库：它把市场营销和客户服务连接起来。

（4）分析能力：CRM的一个重要方面在于它具有使客户价值最大化的分析能力。

#### 2.1.3 供应链管理SCM

SCM理念：强强联合，整合与优化“三流”（信息流-需求信息流、供应信息流，资金流，物流），打通企业间“信息孤岛”，严格的数据交换标准。将制造商、供应商、分销商、零售商，在计划（策略性）、采购、制造、配送、退货等各方面联系起来。

信息流需要进一步了解其分类：

需求信息流：如客户订单、生产计划、采购合同等

供应信息流：如入库单、完工报告单、库存记录、可供销售量、提货发运单等

#### 2.1.4 产品数据管理PDM

（1）PDM理念：用来管理所有与产品相关信息和与产品相关的过程。

（2）PDM发展阶段：配合CAD-->产品数据管理-->产品协同商务。

（3）PDM核心功能：i)数据库和文档管理；ii)产品结构与配置管理；iii)生命周期管理和流程管理；iv)集成开发接口。

#### 2.1.5 商业智能BI

（1）过程

需求分析=>数据仓库建模=>数据抽取=>建立BI分析报表=>用户培训和数据模拟测试=>系统改进和完善

（2）相关技术：数据仓库+数据挖掘+OLAP

数据仓库

数据仓库的特点：面向主题；集成的；相对稳定的（非易失的）；反映历史变化（随着时间变化）。

|  |  |
| --- | --- |
| 数据库 | 数据仓库【特点】 |
| 面向应用：按应用组织数据 | 面向主题：按主题组织数据 |
| 零散的：一个应用对应一个数据库 | 集成的：整个企业对应一个数据仓库 |
| CRUD：增删改查是常态 | 相对稳定的（非易失的）：  查询为主、基本无修改与删除 |
| 解决当下应用问题 | 反映历史变化（时变的）：  各个阶段信息都有，并可做预测未来趋势 |

（3）数据湖

概念：数据湖是一个存储企业的各种各样原始数据的大型仓库，其中的数据可供存取、处理、分析即传输。

特点：数据湖从企业的多个数据源获取原始数据，并且针对不同的目的，同一份原始数据还可能有多种满足特定内部模型格式的数据副本。因此，数据湖中被处理的数据可能是任意类型的信息，从结构化数据到完全非结构化数据。

区别：数据仓库仅支持分析处理，数据湖既支持分析处理，也支持事务处理。

比较

|  |  |  |
| --- | --- | --- |
| 维度 | 数据仓库 | 数据湖 |
| 数据 | 清洗过的数据  结构化的数据 | 原始数据  结构化，半结构化数据 |
| 模式 | 数据存储之前定义数据模式  数据集成之前完成大量工作  数据的价值提前明确 | 数据存储之后定义数据模式  提供敏捷，简单的数据集成  数据的价值尚未明确 |
| 存取方法 | 标准SQL接口 | 应用程序，类SQL的程序 |
| 优势 | 多数据源集成  干净，安全的数据  转换一次，多次使用 | 无限扩展性  并行执行  支持编程框架  数据经济 |

#### 2.1.6 业务流程分析方法

（1）概念

企业业务流程包含三个要素，分别是实体、对象和活动。业务流程发生在实体之间，它们可以是企业间的、功能间的，也可以是人与人之间的；业务流程的功能就是对对象进行操作，这些对象既可以是物理的（例如，订单等），也可以是逻辑的（例如，信息等）；业务流程涉及管理活动和业务操作活动。

（2）分析方法

价值链分析法：找出或设计出那些能够使顾客满意，实现顾客价值最大化的业务流程。

客户关系分析法：把CRM用在业务流程的分析上。

供应链分析法：从企业供应链的角度分析企业的业务流程。

基于ERP的分析法：将企业的业务流程看作是一个紧密联接的供应链，将供应商和企业内部的采购、生产、销售，以及客户紧密联系起来，对供应链上的所有环节进行有效管理，实现对企业的动态控制和各种资源的集成和优化

业务流程重组：通过重新审视企业的价值链，从功能成本的比较分析中，确定企业在哪些环节具有比较优势。

#### 2.1.7 业务流程重组BPR

BPR是对企业的业务流程进行根本性的再思考和彻底性的再设计，从而获得可以用诸如成本、质量、服务和速度等方面的业绩来衡量的显著性的成就。

（1）基本原则：以流程为中心的原则，团队管理原则（以人为本），以客户为导向的原则

（2）过程：项目的启动，拟定变革计划，建立项目团队，分析目标流程，重新设计目标流程，实施新的设计，持续改进，重新开始。

（3）基于BPR的系统规划：战略规划，流程规划，数据规划，功能规划，系统实施。

#### 2.1.8业务流程管理BPM

（1）BPM是一种以规范化的构造端到端的卓越业务流程为中心，以持续的提高组织业务绩效为目的的系统化方法。

（2）PDCA闭环的管理过程：明确业务流程所欲获取的成果，开发和计划系统的方法，实现以上成果；系统地部署方法，确保全面实施；根据对业务的简称和分析以及持续的学习活动，评估和身材所执行的方法。并进一步提出计划和实施改进措施。

（3）流程管理包含三个层面：规范流程、优化流程和再造流程。

（4）BPM与BPR区别

BPM与BPR管理思想最根本的不同就在于流程管理并不要求对所有的流程进行再造。构造卓越的业务流程并不是流程再造，而是根据现有流程的具体情况，对流程进行规范化的设计。

#### 2.1.9电子商务分类

|  |  |
| --- | --- |
| 类型 | 应用 |
| B2B | 1688，慧聪网 |
| C2C | 闲鱼 |
| B2C | 京东，天猫 |
| C2B | 个人给企业提供咨询服务 |
| O2O（Online To Offline）  【线上对线下】 | 团购 |

主要了解电子商务的类型：

企业对消费者（B2C）：京东，当当，天猫，亚马逊。

企业对企业（B2B）：阿里巴巴。

消费者对消费者（C2C）：闲鱼。

线上对线下（O2O）：与前三种类型的分类维度不一样。如：网上南航官方网站购买机票，就既属于B2C，也属于O2O。

### 2.2企业应用集成（⭐⭐⭐）

#### 2.2.1企业集成分类

按组织范围分：

|  |  |  |
| --- | --- | --- |
| 内部集成 | 业务过程的集成 | 实现不同应用系统中的流程能够无缝连接 |
| 应用系统的集成 | 实现不同应用系统之间能够实现数据和方法的共享 |
| 数据的集成 | 实现不同系统的数据交流与共享 |
| 技术平台的集成 | 软件/硬件/网络层面的集成 |
| 外部集成 | 通过门户网站和互联网实现企业内外部信息资源的有效交流和集成 | |
| 通过与合作伙伴信息系统的对接，建立动态的企业联盟 | |

按集成点分：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 集成点 | 效果 | 解题关键点 |
| 界面集成 | 界面 | 统一入口，产生“整体”感觉 | “整体”感觉  最小代价实现一体化操作 |
| 数据集成 | 数据 | 不同来源的数据逻辑或物理上“集中” | 其他集成方法的基础 |
| 控制集成 | 应用逻辑 | 调用其他系统已有方法，达到集成效果 |  |
| 业务流程集成（过程集成） | 应用逻辑 | 跨企业，或优化流程而非直接调用 | 企业之间的信息共享能力 |
| 门户集成 |  | 将内部系统对接到互联网上 | 发布到互联网上 |

（1）表示集成（界面集成）

把各应用系统的界面集成起来，统一入口，产生“整体”感觉。



（2）数据集成

数据集成是应用集成和业务过程集成的基础。把不同来源、格式、特点性质的数据在逻辑上或物理上有机地集中，从而为企业提供全面的数据共享。ETL、数据仓库、联邦数据库都可视为数据集成。



（3）控制集成（功能集成、应用集成、API集成）

业务逻辑层次集成，可以借助于远程过程调用或远程方法调用、面向消息的中间件等技术。



（4）业务流程集成（过程集成）

进行业务流程集成时，企业必须对各种业务信息的交换进行定义、授权和管理，以便改进操作、减少成本、提高响应速度。

按传输方式分：

|  |  |
| --- | --- |
|  | 特点 |
| 消息集成 | 数据量小，交互频繁，立即地，异步 |
| 共享数据库 | 交互频繁，立即地，同步 |
| 文件传输 | 数据量大，交互频度小，即时性要求低（月末，年末） |

（5）消息集成

适用于数据量小、但要求频繁地、立即地、异步地数据交换场合。

（6）共享数据库

实时性强、可以频繁交互，数据的交换属于同步方式。

（7）文件传输

适用于数据量大、交换频度小、即时性要求低的情况。

## 3 章节问答

（1）面向信息集成、面向过程集成、面向服务集成与界面集成、数据集成、应用集成、过程集成有什么区别和联系？

答：

按照集成的层次进行分类的，所以将企业内的集成分为：界面集成、平台集成、数据集成、应用集成、过程集成等。

按照集成技术来分为：面向信息的集成、面向过程的集成和面向服务的集成。（参见教程P420 集成技术的发展展望）。

对于二者的联系： 一般而言，界面集成、平台集成、数据集成、应用集成一般可通过面向信息的集成技术实现，过程集成一般可通过面向过程的集成技术可实现，但其中又常包含接口集成等面向信息的集成技术。当然，应用集成和过程集成也可以通过面向服务的集成技术来实现。

# 第二章 软件工程

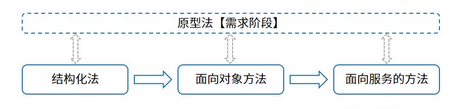
## 1 考情分析

### 1.1 本章概要

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 软件开发 | 软件开发方法（⭐） |
| 2 | 软件开发模型（⭐⭐⭐⭐⭐） |
| 3 | 逆向工程（⭐） |
| 1 | 需求工程 | 概述 |
| 2 | 需求开发（⭐⭐⭐⭐⭐） |
| 3 | 需求管理（⭐⭐⭐） |
| 1 | 系统建模过程 | 系统建模过程 |
| 1 | 系统设计 | 系统设计过程 |
| 2 | 人机界面设计（⭐） |
| 3 | 结构化设计（⭐） |
| 4 | 面向对象设计（⭐⭐⭐⭐⭐） |
| 1 | 软件测试  （⭐⭐⭐⭐⭐） | 概念和目标 |
| 4 | 测试阶段和分类 |
| 5 | 面向对象测试 |
| 1 | 系统运行与软件维护 | 系统转换计划（⭐⭐⭐） |
| 2 | 系统维护 （⭐） |

### 2 考点精讲2.1软件开发

#### 2.1.1 软件开发方法（⭐）



（1）结构化开发方法

用户至上，自顶向下，逐步分解（求解），严格区分工作阶段，每阶段有任务与成果，强调系统开发过程的整体性和全局性，系统开发过程工程化，文档资料标准化。

优点：

理论基础严密，它的指导思想是用户需求在系统建立之前就能被充分了解和理解。由此可见，结构化方法注重开发过程的整体性和全局性。

缺点：

开发周期长；文档、设计说明繁琐，工作效率低；阶段固化，不善变化，适用于需求明确。

（2）原型法开发方法

适用于需求不明确的开发，按功能分-水平原型（界面）、垂直原型（复杂算法），按最终结果分-抛弃式原型、演化式原型。原型法的特点在于原型法对用户的需求是动态响应、逐步纳入的，系统分析、设计与实现都是随着对一个工作模型的不断修改而同时完成的，相互之间并无明显界限，也没有明确分工。

抛弃型原型(Throw-It-Away Prototype)，此类原型在系统真正实现以后就放弃不用了。

进化型原型(Evolutionary Prototype)，此类原型的构造从目标系统的一个或几个基本需求出发，通过修改和追加功能的过程逐渐丰富，演化成最终系统。

（3）面向对象方法

用面向对象方法开发软件，通常需要建立三种形式的模型：对象模型（描述系统数据结构）、动态模型（描述系统控制结构）、功能模型（描述系统功能）。

（4）面向服务的方法

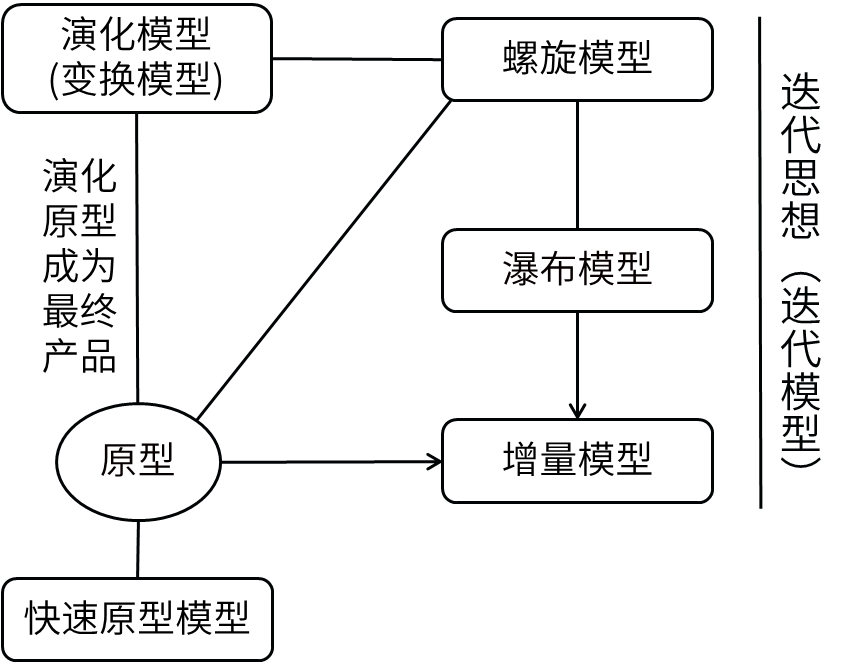
以粗粒度、松散耦合的系统功能为核心，强调系统功能的标准化和构件化，加强了系统的灵活性、可复用性和可演化性。

从概念上讲，SO方法有三个主要的抽象级别：操作、服务、业务流程

（5）其他开发方法

形式化方法（净室软件工程【受控污染级别的环境】；数学模型化；所有东西均可证明/验证，而不是测试）、统一过程方法、敏捷方法、基于架构的开发方法（ABSD）

#### 2.1.2 软件开发模型（⭐⭐⭐⭐⭐）



（1）原型模型

典型的原型开发方法模型。适用于需求不明确的场景，可以帮助用户明确需求。

原型的发展方向:

（2）瀑布模型

瀑布模型是将软件生存周期中的各个活动规定为依线性顺序连接的若干阶段的模型，包括需求分析、设计、编码、运行与维护。

瀑布模型的特点是容易理解，管理成本低，每个阶段都有对应的成果产物，各个阶段有明显的界限划分和顺序要求，一旦发生错误，整个项目推倒重新开始。

适用于需求明确的项目，一般表述为需求明确、或二次开发，或者对于数据处理类型的项目

（3）增量模型

融合了瀑布模型的基本成分和原型实现的迭代特征，可以有多个可用版本的发布，核心功能往往最先完成，在此基础上，每轮迭代会有新的增量发布，核心功能可以得到充分测试。强调每一个增量均发布一个可操作的产品。

（4）螺旋模型

典型特点是引入了风险分析。结合了瀑布模型和演化模型的优点，最主要的特点在于加入了风险分析。它是由制定计划、风险分析、实施工程、客户评估这一循环组成的，它最初从概念项目开始第一个螺旋。

（5）V模型

强调测试贯穿项目始终，而不是集中在测试阶段。是一种测试的开发模型。

（6）喷泉模型

典型的面向对象的模型。特点是迭代、无间隙。会将软件开发划分为多个阶段，但各个阶段无明显界限，并且可以迭代交叉。

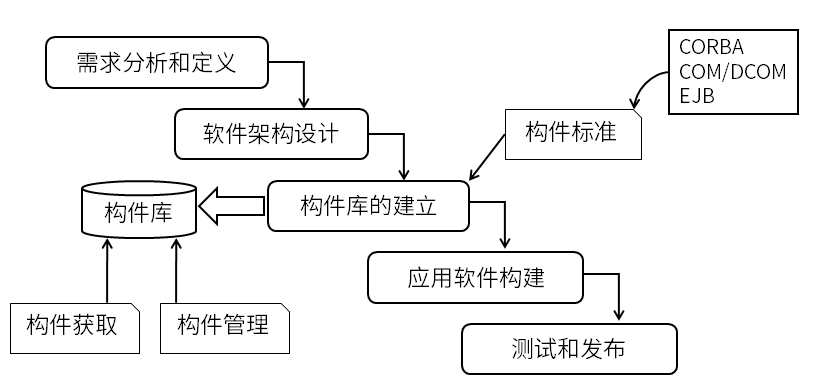
（7）快速应用开发RAD

概念：RAD是瀑布模型的一个高速变种，适用比传统生命周期快得多的开发方法，它强调极短的开发周期，通常适用基于构件的开发方法获得快速开发。

过程：业务建模，数据建模，过程建模，应用生成，测试与交付

适用性：RAD对模块化要求比较高，如果某项功能不能被模块化，则其构件就会出问题；如果高性能是一个指标，且必须通过调整结构使其适应系统构件才能获得，则RAD也有可能不能奏效；RAD要求开发者和客户必须在很短的时间完成一系列的需求分析，任何一方配合不当都会导致失败；RAD只能用于管理信息系统的开发，不适合技术风险很高的情况。

（8）构件组装模型



（9）统一过程（在软考中UP、RUP都指统一过程）

典型特点是用例驱动、以架构为中心、迭代和增量。

统一过程把一个项目分为四个不同的阶段：

构思阶段（初始/初启阶段）：包括用户沟通和计划活动两个方面，强调定义和细化用例，并将其作为主要模型。

细化阶段（精化阶段）：包括用户沟通和建模活动，重点是创建分析和设计模型，强调类的定义和体系结构的表示。

构建阶段：将设计转化为实现，并进行集成和测试。

移交阶段：将产品发布给用户进行测试评价，并收集用户的意见，之后再次进行迭代修改产品使之完善。

（10）敏捷开发敏捷开发是一种以人为核心、迭代、循序渐进的开发方法，适用于小团队和小项目，具有小步快跑的思想。常见的敏捷开发方法有极限编程法、水晶法、并列争球法和自适应

软件开发方法。

极限编程（XP） ：一些对费用控制严格的公司中的使用，非常有效。四大价值观（沟通【加强面对面沟通】、简单【不过度设计】、反馈【及时反馈】、勇气【接受变更的勇气】）十二大最佳实践（简单设计、测试驱动、代码重构、结对编程、持续集成、现场客户、发行版本小型化、系统隐喻、代码集体所有制、规划策略、规范代码、40小时工作机制）。

水晶方法：探索了用最少纪律约束而仍能成功的方法，从而在产出效率与易于运作上达到一种平衡。

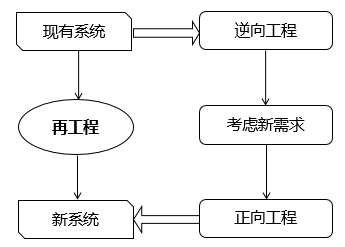
开放式源码：程序开发人员在地域上分布很广【其他方法强调集中办公】。

SCRUM：明确定义了的可重复的方法过程。

功用驱动开发方法（FDD）：编程开发人员分成两类：首席程序员和“类”程序员。

ASD方法：其核心是三个非线性的、重叠的开发阶段：猜测、合作与学习。

#### 2.1.3 逆向工程（⭐）



【逆向工程是设计的恢复过程】

实现级：包括程序的抽象语法树、符号表、过程的设计表示

结构级：包括反映程序分量之间相互依赖关系的信息，例如调用图、结构图、程序和数据结构

功能级：包括反映程序段功能及程序段之间关系的信息，例如数据和控制流模型

领域级：包括反映程序分量或程序诸实体与应用领域概念之间对应关系的信息，例如实体关系模型

与逆向工程相关的概念有重构、设计恢复、再工程和正向工程。

（1）重构（restructuring）。重构是指在同一抽象级别上转换系统描述形式。

（2）设计恢复（design recovery）。设计恢复是指借助工具从已有程序中抽象出有关数据设计、总体结构设计和过程设计等方面的信息。

（3）逆向工程（reverse engineering）：逆向工程是分析程序，力图在比源代码更高抽象层次上建立程序的表示过程，逆向工程是设计的恢复过程。

（4）正向工程（forward engineering）。正向工程是指不仅从现有系统中恢复设计信息，而且使用该信息去改变或重构现有系统，以改善其整体质量。

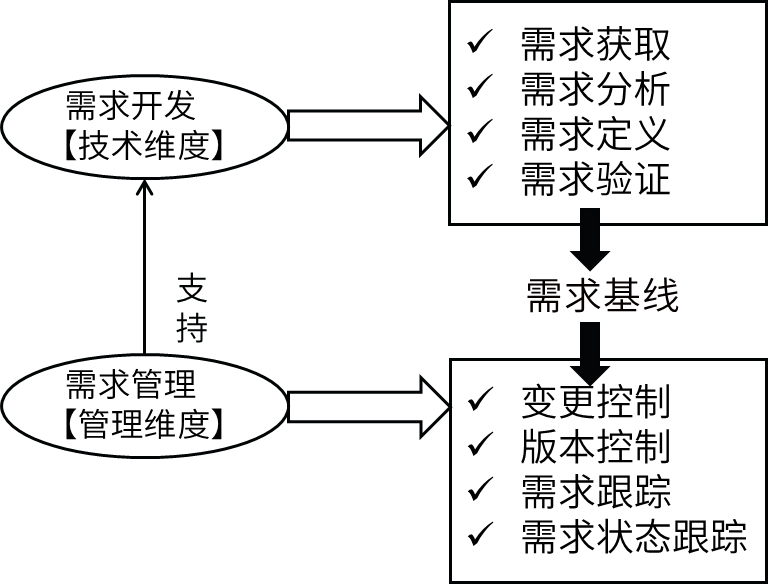
（5）再工程（re-engineering）。再工程是对现有系统的重新开发过程，包括逆向工程、新需求的考虑过程和正向工程三个步骤。

### 2.2 需求工程

#### 2.2.1 概述

软件需求是指用户对系统在功能、行为、性能、设计约束等方面的期望。

软件需求是指用户解决问题或达到目标所需的条件或能力，是系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力，以及反映这些条件或能力的文档说明。



#### 2.2.2 需求开发

##### 2.2.2.1 需求获取

需求分类

【需求的层次分类】

业务需求：是指反应企业或客户对系统高层次的目标要求，通常来自项目投资人、购买产品的客户、客户单位的管理人员、市场营销部门或产品策划部门等。通过业务需求可以确定项目视图和范围，为以后的开发工作奠定了基础。

用户需求：描述的是用户的具体目标，或用户要求系统必须能完成的任务。也就是说，用户需求描述了用户能使用系统来做些什么。

系统需求：是从系统的角度来说明软件的需求，包括功能需求、非功能需求和设计约束等。

功能需求也称为行为需求，它规定了开发人员必须在系统种实现的软件功能，用户利用这些功能来完成任务，满足业务要求。

性能需求（非功能需求）是指系统必须具备的属性或品质，又可细分为软件质量属性和其他非功能需求。

设计约束也称为限制条件或补充规约，通常是对系统的一些约束说明。

【需求的QFD分类】

质量功能部署QFD是一种将用户要求转化成软件需求的技术，其目的是最大限度地提升软件工程过程中用户的满意度。QFD将软件需求分为三类：

常规需求（基本需求）：用户认为系统应该做到的功能或性能，实现越多用户会越满意。

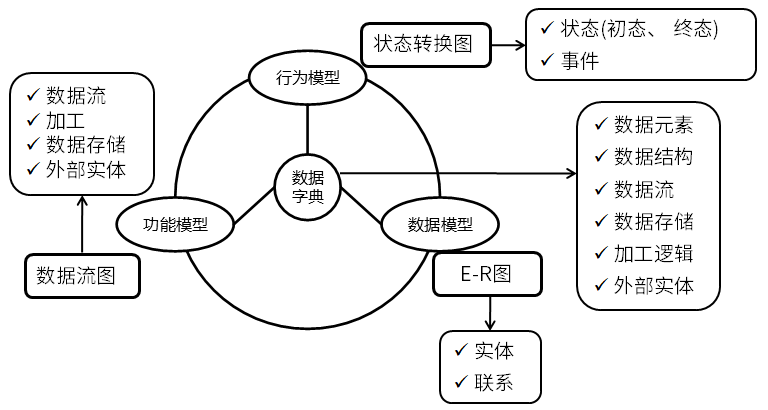
期望需求：用户想当然认为系统应具备的功能或性能，但并不能正确描述自己想要得到的这些功能或性能需求。如果期望需求没有得到实现，会让用户感到不满意。

兴奋需求（意外需求）：是用户要求范围外的功能或性能，实现这些需求用户会更高兴，但不实现也不影响其购买的决策。

##### 2.2.2.2 需求分析

结构化需求分析（SA）（⭐⭐）

（1）结构化分析过程



（2）结构化分析工具-数据流图DFD

|  |  |  |
| --- | --- | --- |
| 元素 | 说明 | 图元 |
| 数据流 | 由一组固定成分的数据组成，表示数据的流向。每个数据流通常有一个合适的名词，反映数据流的含义 |  |
| 加工 | 加工描述了输入数据流到输出数据流之间的变换，也就是输入数据流做了什么处理后变成了输出数据流 |  |
| 数据存储  （文件） | 用来表示暂时存储的数据，每个文件都有名字。流向文件的数据流表示写文件，流出的表示读文件 |  |
| 外部实体 | 指存在于软件系统外的人员或组织 |  |

（3）结构化需求分析-流程分析STD-状态转换图



面向对象需求分析（⭐⭐⭐⭐⭐）

（1）面向对象基本概念

对象：属性（数据）+方法（操作）+对象ID

类（实体类/控制类/边界类）

实体类映射需求中的每个实体，实体类保存需要存储在永久存储体中的信息，例如，在线教育平台系统可以提取出学员类和课程类，它们都属于实体类。

控制类是用于控制用例工作的类，一般是由动宾结构的短语（“动词+名词”或“名词+动词”）转化来的名词，例如，用例“身份验证”可以对应于一个控制类“身份验证器”，它提供了与身份验证相关的所有操作。

边界类用于封装在用例内、外流动的信息或数据流。边界类位于系统与外界的交接处，包括所有窗体、报表、打印机和扫描仪等硬件的接口，以及与其他系统的接口。

继承与泛化：复用机制

封装：隐藏对象的属性和实现细节,仅对外公开接口

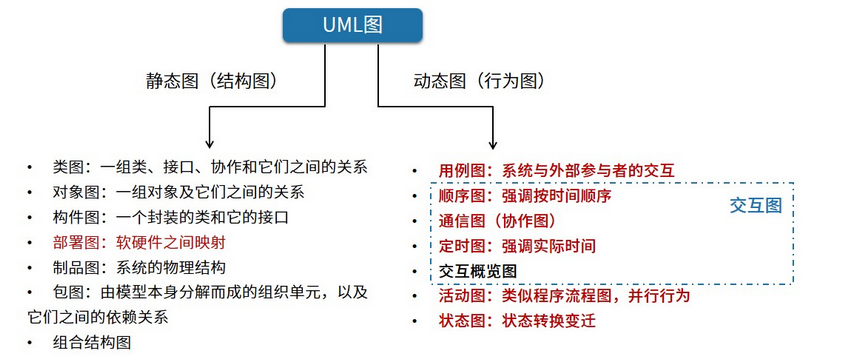
多态：不同对象收到同样的消息产生不同的结果

接口：一种特殊的类，他只有方法定义没有实现

重载：一个类可以有多个同名而参数类型不同的方法

消息和消息通信：消息是异步通信的

（2）UML图分类



（3）UML图关系

用例关系包括：包含关系、扩展关系、泛化关系。

包含关系：其中这个提取出来的公共用例称为抽象用例，而把原始用例称为基本用例或基础用例系：当可以从两个或两个以上的用例中提取公共行为时，应该使用包含关系来表示它们。

扩展关系：如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种分支，则可以将这个用例分为一个基本用例和一个或多个扩展用例，这样使描述可能更加清晰。

泛化关系：当多个用例共同拥有一种类似的结构和行为的时候，可以将它们的共性抽象成为父用例，其他的用例作为泛化关系中的子用例。在用例的泛化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系。

类图/对象图关系：

依赖关系：一个事物发生变化影响另一个事物。

泛化关系：特殊/一般关系

关联关系：描述了一组链，链是对象之间的连接。

聚合关系：整体与部分生命周期不同。

组合关系：整体与部分生命周期相同。

实现关系：接口与类之间的关系

（4）“4+1”视图（⭐⭐）



UML采用4+1视图来描述软件和软件开发过程：

逻辑视图：以问题域的语汇组成的类和对象集合。

进程视图：可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例，描绘了所设计的并发与同步结构。

实现视图：对组成基于系统的物理代码的文件和组件进行建模。

部署视图：把构件部署到一组物理的、可计算的节点上，表示软件到硬件的映射及分布结构。

用例视图：最基本的需求分析模型。

#### 2.2.3 需求管理（⭐⭐⭐）

（1）定义需求基线



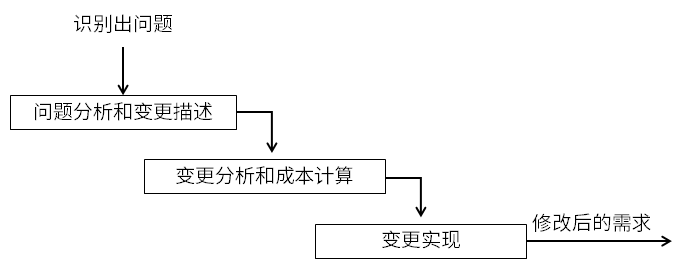
**需求的状态**

（2）需求跟踪

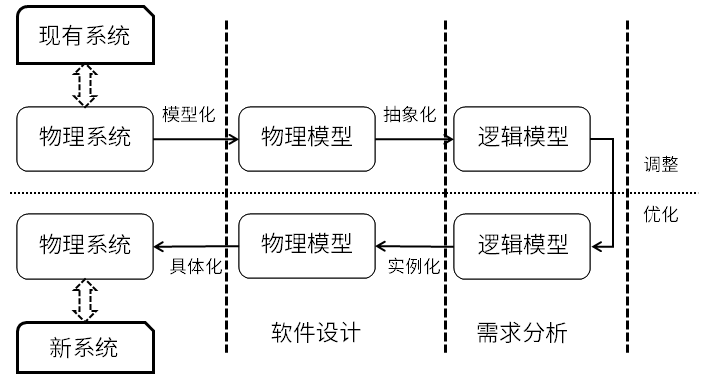


（3）变更控制

带有风险的做法：无足够用户参与，忽略了用户分类，用户需求的不断增加，模棱两可的需求，不必要的特性，过于精简的SRS，不准确的估算。



### 2.3系统建模过程



（1）结构化建模方法：结构化建模方法是以过程为中心的技术，可用于分析一个现有的系统以及定义新系统的业务需求。结构化建模方法所绘制的模型称为数据流图（DFD）。对于流程较为稳定的系统可考虑结构化建模方法。

（2）信息工程建模方法（或数据库建模方法）：信息工程建模方法是一种以数据为中心，但过程敏感的技术，它强调在分析和研究过程需求之前，首先研究和分析数据需求。信息工程建模方法所创建的模型被称为实体-联系图（ERD）。主要用于数据建模。

（3）面向对象建模方法：面向对象建模方法将“数据”和“过程”集成到被称为“对象”的结构中，消除了数据和过程的人为分离现象。面向对象建模方法所创建的模型被称为对象模型。随着面向对象技术的不断发展和应用，形成了面向对象的建模标准，即UML（统一建模语言）。UML定义了几种不同类型的模型图，这些模型图以对象的形式共建一个信息系统或应用系统。目前比较常用的建模方法。

### 2.4系统设计

#### 2.4.1 系统设计过程

（1）概要设计

总体设计

总体布局设计

网络拓扑结构设计

资源配置设计

模块化结构设计

划分功能模块

模块功能和职责

模块间的调用关系

模块间的信息传递

（2）详细设计

代码设计：是信息分类和编码的工作，是将系统中有某些共同属性或特征的信息归并在一起，并利用便于计算机和人识别和处理的符号来表示这些信息的设计工作。

数据库设计：构建既能客观、准确地反映外部世界，又便于人类大脑认识的概念模型，并在此基础上对数据进行建模，转化为数据库管理系统所支持的数据模型；选择合适的存储结构和存储方法，最终完成数据库的设计工作。

输入/输出设计：输入/输出设计主要是对以记录为单位的各种输入输出报表格式的描述。另外，对人机对话格式的设计和输入输出装置的选择也在这一步完成。

用户界面设计：用户界面设计是指用户与系统之间架起一座桥梁，主要内容包括：定义界面形式、定义基本的交互控制形成、定义图形和符号、定义通用的功能键和组合键的含义及其操作内容、定义帮助策略等。

处理过程设计：定义每个模块的内部执行过程，包括数据的组织、控制流、每一步的具体加工要求和实施细节。

（3）设计任务

结构设计：定义软件系统各主要部件之间的关系。

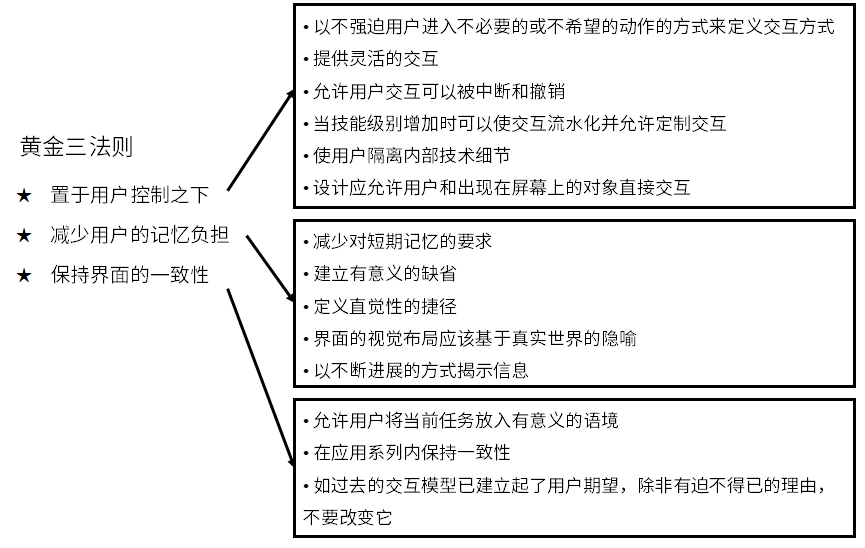
数据设计：将模型转换成数据结构的定义。好的数据设计将改善程序结构和模块划分，降低过程复杂性。

接口设计（人机界面设计）：软件内部，软件和操作系统间以及软件和人之间如何通信。

过程设计：系统结构部件转换成软件的过程描述。

#### 2.4.2 人机界面设计（⭐）

用户界面设计是指用户与系统之间架起一座桥梁，主要内容包括：定义界面形式、定义基本的交互控制形成、定义图形和符号、定义通用的功能键和组合键的含义及其操作内容、定义帮助策略等。



#### 2.4.3 结构化设计（⭐）

（1）特点：抽象化、自顶而下、逐步求精、信息隐蔽、模块独立（高内聚、低耦合）

（2）模块四要素：

输入和输出。模块的输入来源和输出去向都是同一个调用者，即一个模块从调用者那儿取得输入，进行加工后再把输出返回调用者。

处理功能。指模块把输入转换成输出所做的工作。

内部数据。指仅供该模块本身引用的数据。

程序代码。指用来实现模块功能的程序。

（3）功能模块设计的原则

高内聚低耦合

系统分解有层次

较小的数据冗余

（4）模块独立性的度量

聚合：衡量模块内部各元素结合的紧密程度

偶然聚合：模块完成的动作之间没有任何关系，或者仅仅是一种非常松散的关系。

逻辑聚合：模块内部的各个组成在逻辑上具有相似的处理动作，但功能用途上彼此无关。

时间聚合：模块内部的各个组成部分所包含的处理动作必须在同一时间内执行。

过程聚合：模块内部各个组成部分所要完成的动作虽然没有关系，但必须按特定的次序执行。

通信聚合：模块的各个组成部分所完成的动作都使用了同一个数据或产生同一输出数据。

顺序聚合：模块内部的各个部分，前一部分处理动作的最后输出是后一部分处理动作的输入。

功能聚合：模块内部各个部分全部属于一个整体，并执行同一功能，且各部分对实现该功能都比不可少

耦合：度量不同模块间互相依赖的程度

非直接耦合：两个模块之间没有直接关系，它们的联系完全是通过主模块的控制和调用来实现的。

数据耦合：两个模块彼此间通过数据参数交换信息。

标记耦合：一组模块通过参数表传递记录信息，这个记录是某一个数据结构的子结构，而不是简单变量。

控制耦合：两个模块彼此间传递的信息中有控制信息。

外部耦合：一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息。

公共耦合：两个模块之间通过一个公共的数据区域传递信息。

内容耦合：一个模块需要涉及到另一个模块的内部信息。

#### 2.4.4 面向对象设计（⭐⭐⭐⭐⭐）

过程



设计原则

单一职责原则：设计目的单一的类

开放-封闭原则：对扩展开放，对修改封闭

李氏(Liskov)替换原则：子类可以替换父类

依赖倒置原则：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程

接口隔离原则：使用多个专门的接口比使用单一的总接口要好

组合重用原则：要尽量使用组合，而不是继承关系达到重用目的

迪米特(Demeter)原则(最少知识法则)：一个对象应当对其他对象有尽可能少的了解

设计模式

概念

架构模式：软件设计中的高层决策，例如C/S结构就属于架构模式，架构模式反映了开发软件系统过程中所作的基本设计决策

设计模式：主要关注软件系统的设计，与具体的实现语言无关

惯用法：是最低层的模式，关注软件系统的设计与实现，实现时通过某种特定的程序设计语言来描述构件与构件之间的关系。每种编程语言都有它自己特定的模式，即语言的惯用法。例如引用-计数就是C++语言中的一种惯用法

设计模式适用场景

创建型模式

|  |  |  |
| --- | --- | --- |
| 设计模式名称 | 简要说明 | 速记关键字 |
| Abstract Factory  抽象工厂模式 | 提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类 | 生产成系列对象 |
| Builder  构建器模式 | 将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示 | 复杂对象构造 |
| Factory Method  工厂方法模式 | 定义一个创建对象的接口，但由子类决定需要实例化哪一个类。 工厂方法使得子类实例化的过程推迟 | 动态生产对象 |
| Prototype  原型模式 | 用原型实例指定创建对象的类型，并且通过拷贝这个原型来创建新的对象 | 克隆对象 |
| Singleton  单例模式 | 保证一个类只有一个实例，并提供一个访问它的全局访问点 | 单实例 |

结构型模式

|  |  |  |
| --- | --- | --- |
| 设计模式名称 | 简要说明 | 速记关键字 |
| Adapter  适配器模式 | 将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作 | 转换接口 |
| Bridge  桥接模式 | 将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化 | 继承树拆分 |
| Composite  组合模式 | 将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性 | 树形目录结构 |
| Decorator  装饰模式 | 动态地给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活 | 附加职责 |
| Facade  外观模式 | 定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用 | 对外统一接口 |
| Flyweight  享元模式 | 提供支持大量细粒度对象共享的有效方法 | 文章共享文字对象 |
| Proxy  代理模式 | 为其他对象提供一种代理以控制这个对象的访问 | 快捷方式 |

行为型模式

|  |  |  |
| --- | --- | --- |
| 设计模式名称 | 简要说明 | 速记关键字 |
| Chain of  Responsibility  职责链模式 | 通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求 | 传递职责 |
| Command  命令模式 | 将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作 | 日志记录，可撤销 |
| Interpreter  解释器模式 | 给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子 | 虚拟机的机制 |
| Iterator  迭代器模式 | 提供一种方法来顺序访问一个聚合对象中的各个元素，而不需要暴露该对象的内部表示 | 数据库数据集 |
| Mediator  中介者模式 | 用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立地改变对象间的交互 | 不直接引用 |
| Memento  备忘录模式 | 在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态 | 可恢复 |
| Observer  观察者模式 | 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新 | 联动 |
| State  状态模式 | 允许一个对象在其内部状态改变时改变它的行为 | 状态变成类 |
| Strategy  策略模式 | 定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化 | 多方案切换 |
| Template Method  模板方法模式 | 定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤 | 文档模板填空 |
| Visitor  访问者模式 | 表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作 | 数据与操作分离 |

### 2.5软件测试（⭐⭐⭐⭐⭐）

#### 2.5.1 概念和目标

（1）概念：为了发现错误而执行程序的过程

（2）目标（《软件测试的艺术》）

是为了发现错误而执行程序的过程

好的测试方案能够发现迄今为止尚未发现的错误

成功的测试将发现至今尚未发现的错误

#### 2.5.2 测试阶段和分类

（1）单元测试

概念：又称模块测试，是通过对每个最小的软件模块进行测试，对照模块的功能说明，检查各个程序模块是否正确地实现了规定的功能，确保其能正常工作。

测试主体：单元测试由开发人员执行

测试内容：模块接口测试、局部数据结构测试、路径测试、错误处理测试、边界测试（模块测试，模块功能、性能、接口等）。

（2）集成测试

概念：在单元测试的基础上，需要将所有模块按照概要设计说明书和详细设计说明书的要求进行组装。主要目的是验证组成软件系统的各模块的接口和交换作用。（模块间的接口）

组装时需要考虑的问题：在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失

一个模块的功能是否会对另一个模块的功能产生不利的影响；各个子功能组合起来，能否达到预期要求的父功能；全局数据结构是否有问题；单个模块的误差累积起来，是否会放大，以致达到不能接受的程度

模块组装方式



一次性组装方式：其结果：发现有错误，却茫然找不到原因；查错和改错都会遇到困难。

自顶向下的增殖方式：优点：在测试过程中较早地验证了主要的控制和判断点；功能可行性较早地得到证实，还能增强开发者和用户成功的信心。缺点：导致过多的回归测试；增加建桩模块的复杂度，导致增加附加测试

自底向上的增殖方式：优点：可以把容易出问题的部分在早期解决；缺点：对主要的控制直到最后才接触到；可以实施多个模块的并行测试，提高测试效率。

混合增殖方式，完成的标志：成功地执行了测试计划中规定的所有集成测试；修正了所发现的错误（与用户就剩余错误的修改计划达成一致）；测试结果通过了专门小组的评审

（3）“冒烟测试”（英文：smoke testing）这一术语源自硬件行业。对一个硬件或硬件组件进行更改或修复后，直接给设备加电。如果没有冒烟，则该组件就通过了测试。

（4）确认测试

概念：又称合格性测试，用来检验软件是否符合用户的需求。一般采用黑盒测试法，通过一系列证明软件功能和要求的测试来实现。确认测试着重考虑软件是否满足合同规定的所有功能和性能、文档资料是否完整。确认人机界面和其他方面（如可移植性、兼容性、错误恢复能力和可维护性等）是否令用户满意确认测试过程的重要环节就是配置审查工作。配置审查的文件资料包括用户手册、操作手册和设计资料。其目的在于确保软件的所有文件资料均已编写齐全，用于支持日后软件的维护工作。（验证软件与需求的一致性。内部确认测试、Alpha测试、Beta测试，验收测试）

（5）系统测试

概念：系统测试真实环境下，验证完整的软件配置项能否和系统正确连接。将软件与整个系统的硬件、外设、支持软件、数据和人员等结合起来，以需求规格说明为依据，在实际运行环境下进行测试。检验其是否有不符合系统说明书的地方。

系统测试过程分为计划与准备、执行、返工与回归测试3个阶段

内容：系统测试一般要完成功能测试、性能测试、恢复测试、安全测试、强度测试以及其他限制条件的测试。

（6）验收测试

概念：在测试组的协调下，由用户代表执行。检验系统说明书的各项功能与性能是否实现和满足要求。验收测试完全采用黑盒测试技术，其主要任务是文档资料的审查验收、软件系统的功能测试、性能测试、强化测试、性能降级执行方式测试、检查系统的余量要求、安装测试以及用户操作测试。

（7）回归测试

测试软件变更之后，变更部分的正确性对变更需求的符合性。

（8）α与β测试

Alpha测试：是在开发环境下进行的测试，由用户/内部用户模拟实际操作环境下进行的受控测试。目的是评价软件产品的功能、可使用性、可靠性、性能和支持。尤其要注重产品的界面和特色。

Beta测试：是用户在实际使用环境下进行的测试。

（9）白盒测试

概念：根据内部结构和逻辑来设计测试用例，对程序路径和过程进行测试。

#### 2.5.3 面向对象的测试

算法层（单元测试）：包括等价类划分测试、组合功能测试（基于判定表的测试）、递归函数测试和多态消息测试。（方法层次）

类层（模块测试）：包括不变式边界测试、模态类测试和非模态类测试

模板层/类树层（集成测试）：包括多态服务测试和展平测试

系统层（系统测试）

### 2.6系统运行与软件维护

#### 2.6.1系统转换计划（⭐⭐⭐）

（1）遗留系统演化策略



淘汰策略：遗留系统的技术含量较低，且具有较低的业务价值。对遗留系统的完全淘汰是企业资源的根本浪费，系统分析师应该善于“变废为宝”，通过对遗留系统功能的理解和借鉴，可以帮助新系统的设计，降低新系统开发的风险。

继承策略：遗留系统的技术含量较低，已经满足企业运作的功能或性能要求，但具有较高的商业价值，目前企业的业务尚紧密依赖该系统。对这种遗留系统的演化策略为继承。在开发新系统时，需要完全兼容遗留系统的功能模型和数据模型。为了保证业务的连续性，新老系统必须并行运行一段时间，再逐渐切换到新系统上运行。

改造策略：遗留系统具有较高的业务价值，基本上能够满足企业业务运作和决策支持的需要。这种系统可能建成的时间还很短，对这种遗留系统的演化策略为改造。改造包括系统功能的增强和数据模型的改造两个方面。系统功能的增强是指在原有系统的基础上增加新的应用要求，对遗留系统本身不做改变；数据模型的改造是指将遗留系统的旧的数据模型向新的数据模型的转化。

集成策略：遗留系统的技术含量较高，但其业务价值较低，可能只完成某个部门（或子公司）的业务管理。这种系统在各自的局部领域里工作良好，但对于整个企业来说，存在多个这样的系统，不同的系统基于不同的平台、不同的数据模型，形成了一个个信息孤岛，对这种遗留系统的演化策略为集成。

（2）新旧系统转换策略

直接转换：接转换是在原有系统停止运行的某一时刻，新系统立即投入运行，中间没有过度阶段。采用这种方式时，人力和费用最省，适用于系统不太复杂或现有系统完全不能使用的场合。但是这种方式风险高。

并行转换：并行转换就是新系统和旧系统并行工作一段时间，经过这段时间的试运行后，再用新系统正式替换下现有系统。那么这种方式，它的好处就是风险很小。在转换期间还可以同时比较新旧两套系统的性能，而且能够让操作人员得到全面的培训，所以对于一些比较大的信息系统，或者处理过程比较复杂，数据比较重要的系统。并行转换是一种最常用的转换方式。那么这种转换方式也有缺点，缺点就在于两套系统并行期间。要有两套人马，或者两套处理方式同时并存，在人力和费用消耗比较大，转换的周期比较长，而且难以控制新旧系统当中数据的变化。所以这就要求要做好转换计划，并且要加强管理。

分段转换：这是直接转换和并行转换的接合，也就是分期分批、逐步转换。一般比较大的系统可以采用这种方式比较合适，他能够保证软件平稳运行，费用也不太高，就是将大的系统分成多个子系统，每成熟一个子系统就切换一个子系统，主要是分期分批。这种分段转换的策略，它的优点就是成熟一个子系统就转换一个子系统。这种新旧转换，震动比较小，用户比较容易接受。但是由于采取的是渐进的方式，会导致新旧系统的转换周期比较长。

#### 2.6.2 系统维护 （⭐）

（1）概念

软件维护是生命周期的一个完整部分。可以将软件维护定义为需要提供软件支持的全部活动，这些活动包括在交付前完成的活动，以及交付后完成的活动。交付前完成的活动包括交付后运行的计划和维护计划等；交付后的活动包括软件修改、培训、帮助资料等。

（2）分类

正确性维护：指改正在系统开发阶段已发生而系统测试阶段尚未发现的错误。

适应性维护：指使应用软件适应环境变化【外部环境、数据环境】而进行的修改。

完善性维护：扩充功能和改善性能而进行的修改。

预防性维护：为了适应未来的软硬件环境的变化，应主动增加预防性的新的功能，以使用系统适应各类变化而不被淘汰。如将专用报表功能改成通用报表生成功能，以适应将来报表格式的变化。

## 3 章节问答

（1）演化模型跟螺旋模型，增量模型，原型法开发的关系？

答：

首先是原型化，原型化又可以分为两种，抛弃式原型和演化式原型，其中 最终演化成一个产品的就演变成为演化模型。螺旋模型式是通过演化模型和瀑布模型的结合所产生的，并且螺旋模型强调了风险管理。增量模型是原型模型和瀑布模型所结合而产生的。其中螺旋模型和增量模型的区别：螺旋模型他是一直旋下去，旋到最后才是一个产品，而增量模型每一个增量都发布了一个可操作的产品，这是它的一个细微的区别

（2）面向对象方法中的分析模型、设计模型有哪些？？

答：

面向对象的分析模型主要由顶层架构图、用例与用例图、领域概念模型构成；设计模型则包含以包图表示的软件体系结构图、以交互图表示的用例实现图、完整精确的类图、针对复杂对象的状态图和用以描述流程化处理过程的活动图等。

（3）结构化方法中的设计模型有哪些？

答：

结构化设计包括了概要设计和详细设计。模型主要是用数据流图表示功能模型，用状态转换图表示行为模型，用E-R图表示数据模型。

（4）自顶向下开发方法的优缺点。

答：

自顶向下方法的优点是：可为企业或机构的重要决策和任务实现提供信息；支持企业信息系统的整体性规划，并对系统的各子系统的协调和通信提供保证；方法的实践有利于提高企业人员整体观察问题的能力，从而有利于寻找到改进企业组织的途径。

自顶向下方法的缺点是：对系统分析和设计人员的要求较高；开发周期长，系统复杂，一般属于一种高成本、大投资的工程；对于大系统而言自上而下的规划对于下层系统的实施往往缺乏约束力；从经济角度来看，很难说自顶向下的做法在经济上是合算的。

（5）为什么自顶向下的单元测试中不需要另外编写驱动模块？

答：

对于模块的划分，上层为驱动模块（驱动模块用来调用被测模块），下层为桩模块（桩模块用来模拟被测模块所调用的子模块）。自顶向下测试时，调用模块已经写完，不需要另写驱动模块，但是需要用桩模块来帮助测试。自底向上测试时，底层模块已经写完，不需要另写桩模块，但是需要用驱动模块来帮助测试。

（6）构件模型在使用上的局限性和缺点有哪些？

答：

优点：这大大提高了软件开发的效率，而且构件模型允许多个项目同时开发，降低了费用，提高了可维护性。

缺点：由于存在多种构件标准，缺乏通用的构件组装结构标准，如果自行定义标准的话，会引入较大的风险；构件可重用性和软件系统高效性之间不易协调，需要权衡。

（7）设计模式要求掌握到什么程度？

答：

设计模式有一定难度。对于设计模式要求掌握如下内容：

1、设计模型三种类型的定位；

2、设计模式分类；

3、各设计模式的应用场景及特点；

可见上文列出的关键字表格进行记忆。

# 第三章 软件架构设计

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 软件架构的概念  （⭐） | 什么是架构（暂无定论） |
| 1 | 软件结构风格  （⭐⭐⭐⭐⭐） | 数据流风格 |
| 2 | 调用/返回风格 |
| 3 | 独立构件风格 |
| 4 | 虚拟机风格 |
| 5 | 仓库风格（以数据为中心的风格） |
| 6 | 闭环控制架构（过程控制） |
| 1 | 典型架构应用 | 层次架构 |
| 3 | 基于服务的架构（SOA） |
| 4 | 微服务-混合风格 |
| 1 | 特定领域软件架构（DSSA）  （⭐⭐⭐⭐） | 基本活动 |
| 2 | 领域分析机制 |
| 1 | 基于架构的软件开发方法（⭐⭐⭐⭐） | 基于架构的软件开发方法 |
| 1 | 架构评估  （⭐⭐⭐⭐⭐） | 为什么要进行架构评估呢？ |
| 2 | 架构评估到底评什么？ |
| 3 | 架构设计重点关注非功能设计（质量属性）（⭐⭐⭐⭐⭐） |
| 4 | 软件架构评估方法（⭐⭐⭐） |
| 1 | 产品线 | 基本概念 |
| 2 | 特点 |
| 3 | 过程模型 |
| 4 | 建立方式 |
| 5 | 组织结构类型 |
| 6 | 成功实施产品线主要取决因素 |
| 1 | WEB设计 | 集群 |
| 2 | 负载均衡技术 |
| 3 | 有状态和无状态问题 |
| 4 | CDN内容分发网络 |
| 5 | XML与JSON |
| 6 | 缓存 |
| 7 | REST |

## 2 考点精讲

### 2.1 软件架构的概念（⭐）

#### 2.1.1 什么是架构（暂无定论）

架构设计就是需求分配，即将满足需求的职责分配到组件上。

软件架构风格是描述某一特定应用领域中系统组织方式的惯用模式。架构风格定义一个系统家族，即一个体系结构定义一个词汇表和一组约束。词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。

软件架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、指导元素集成的模式以及这些模式的约束组成。

架构的作用：

软件架构是项目干系人进行交流的手段，明确了对系统实现的约束条件，决定了开发和维护组织的组织结构，制约着系统的质量属性

软件架构使推理和控制的更改更加简单，有助于循序渐进的原型设计，可以作为培训的基础

软件架构是可传递和可复用的模型，通过研究软件架构可能预测软件的质量

### 2.2 软件架构风格（⭐⭐⭐⭐⭐）

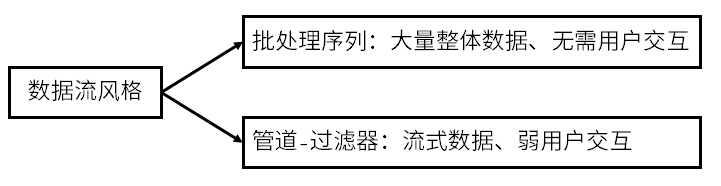
架构设计的一个核心问题是能否达到架构级的软件复用

架构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个构件有效地组织成一个完整的系统

架构风格定义了用于描述系统的术语表和一组指导构建系统的规则

|  |  |
| --- | --- |
| 五大架构风格 | 子风格 |
| 数据流风格 | 批处理、管道-过滤器 |
| 调用/返回风格 | 主程序/子程序、面向对象、层次结构 |
| 独立构件风格 | 进程通信、事件驱动系统（隐式调用） |
| 虚拟机风格 | 解释器、规则系统 |
| 仓库风格 | 数据库系统、黑板系统、超文本系统 |

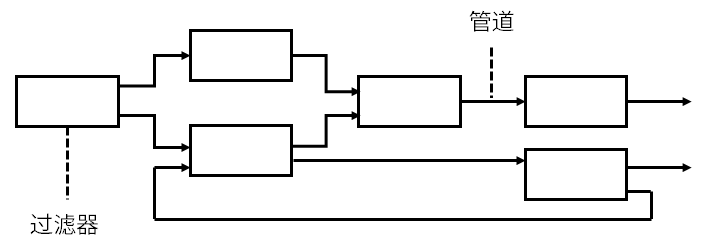
#### 2.2.1 数据流风格



（1）批处理序列

构件为一系列固定顺序的计算单元，构件之间只通过数据传递交互。每个处理步骤是一个独立的程序，每一步必须在其前一步结束后才能开始，数据必须是完整的，以整体的方式传递。

（2）管道-过滤器



每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常是通过对输入数据流的变换或计算来完成的，包括通过计算和增加信息以丰富数据、通过浓缩和删除以精简数据、通过改变记录方式以转化数据和递增地转化数据等。这里的构件称为过滤器，连接件就是数据流传输的管道，将一个过滤器的输出传到另一个过滤器的输入。

早期编译器就是采用的这种架构。要一步一步处理的，均可考虑采用此架构风格。

|  |  |  |
| --- | --- | --- |
| 优点 | 缺点 | 典型实例 |
| 1、松耦合【高内聚-低耦合】；  2、良好的重用性/可维护性； 3、可扩展性【标准接口适配】；  4、良好的隐蔽性  5、支持并行。 | 1、交互性较差； 2、复杂性较高； 3、性能较差（每个过滤器都需要解析与合成数据）。 | 传统编译器  网络报文处理 |

#### 2.2.2 调用/返回风格

（1）主程序/子程序

单线程控制，把问题划分为若干个处理步骤，构件即为主程序和子程序，子程序通常可合成为模块。过程调用作为交互机制，即充当连接件的角色。调用关系具有层次性，其语义逻辑表现为主程序的正确性取决于它调用的子程序的正确性

（2）面向对象

构件是对象，对象是抽象数据类型的实例。在抽象数据类型中，数据的表示和它们的相应操作被封装起来，对象的行为体现在其接受和请求的动作。连接件即是对象间交互的方式，对象是通过函数和过程的调用来交互的

（3）层次结构

概念：构件组织成一个层次结构，连接件通过决定层间如何交互的协议来定义。每层为上一层提供服务，使用下一层的服务，只能见到与自己邻接的层。通过层次结构，可以将大的问题分解为若干个渐进的小问题逐步解决，可以隐藏问题的复杂度。修改某一层，最多影响其相邻的两层（通常只能影响上层）

|  |  |  |
| --- | --- | --- |
| 优点 | 缺点 | 特点 |
| 1、良好的重用性，只要接口不变可用在其它处； 2、可维护性好； 3、可扩展性好，支持递增设计。 | 1、并不是每个系统都方便分层； 2、很难找到一个合适的、正确的层次抽象方法。 3、不同层次之间耦合度高的系统很难实现。 | 各个层次的组件形成不同功能级别的虚拟机；  多层相互协同工作，而且实现透明 |

#### 2.2.3 独立构件风格

|  |  |  |
| --- | --- | --- |
| 优点 | 缺点 | 特点 |
| 1、松耦合。  2、良好的重用性/可修改性/可扩展性。 | 1、构件放弃了对系统计算的控制。一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的过程，它也不能保证这些过程被调用的顺序。 2、数据交换的问题。 3、既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理就存在问题。 | 系统由若干子系统构成且成为一个整体；系统有统一的目标；子系统有主从之分；每一子系统有自己的事件收集和处理机制 |

（1）进程通信

构件是独立的过程，连接件是消息传递。构件通常是命名过程，消息传递的方式可以是点对点、异步或同步方式，以及远程过程（方法）调用等。

（2）事件驱动系统（隐式调用）

构件不直接调用一个过程，而是触发或广播一个或多个事件。构件中的过程在一个或多个事件中注册，当某个事件被触发时，系统自动调用在这个事件中注册的所有过程。一个事件的触发就导致了另一个模块中的过程调用。这种风格中的构件是匿名的过程，它们之间交互的连接件往往是以过程之间的隐式调用来实现的。主要优点是为软件复用提供了强大的支持，为构件的维护和演化带来了方便；其缺点是构件放弃了对系统计算的控制。

#### 2.2.4 虚拟机风格

（1）解释器

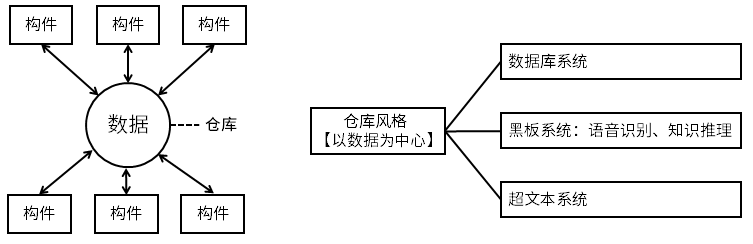
解释器通常包括一个完成解释工作的解释引擎、一个包含将被解释的代码的存储区、一个记录解释引擎当前工作状态的数据结构，以及一个记录源代码被解释执行的进度的数据结构。具有解释器风格的软件中含有一个虚拟机，可以仿真硬件的执行过程和一些关键应用，其缺点是执行效率比较低。

（2）基于规则的系统

基于规则的系统包括规则集、规则解释器、规则/数据选择器和工作内存，一般用在人工智能领域和DSS中。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 子分类 | 优点 | 缺点 | 特点 | 适合领域 |
| 解释器 | 可以灵活应对自定义场景 | 复杂度较高 |  | 适用于需要“ 自定义规则”的场合 |
| 规则为中心 | 在解释器的基础上增加经验规则 | 适用于专家系统 |

#### 2.2.5 仓库风格（以数据为中心的风格）



（1）数据库系统

构件主要有两大类，一类是中央共享数据源，保存当前系统的数据状态；另一类是多个独立处理单元，处理单元对数据元素进行操作。

（2）黑板系统



包括知识源、黑板和控制三部分。知识源包括若干独立计算的不同单元，提供解决问题的知识。知识源响应黑板的变化，也只修改黑板；黑板是一个全局数据库，包含问题域解空间的全部状态，是知识源相互作用的唯一媒介；知识源响应是通过黑板状态的变化来控制的。黑板系统通常应用在对于解决问题没有确定性算法的软件中（信号处理、问题规划和编译器优化等）。

（3）超文本系统

构件以网状链接方式相互连接，用户可以在构件之间进行按照人类的联想思维方式任意跳转到相关构件。超文本是一种非线性的网状信息组织方法，它以结点为基本单位，链作为结点之间的联想式关联。超文本系统通常应用在互联网领域。

现代集成编译环境一般采用这种架构风格。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 架构风格 | 子分类 | 优点 | 缺点 | 特点 | 典型实例 |
| 仓库风格 | 数据库系统 |  |  | 以数据为中心 |  |
| 黑板系统 | 可更改性和可维护性；可重用的知识源；容错性和健壮性 | 测试困难；不能保证有好的解决方案；难以建立好的控制策略；低效；开发困难；缺少并行机制。 | 在以数据为中心的基础上，使用中心数据触发业务逻辑部件 | 语音识别  模式识别  图像处理  知识推理 |

#### 2.2.6 闭环控制架构（过程控制）



当软件被用来操作一个物理系统时，软件与硬件之间可以粗略地表示为一个反馈循环，这个反馈循环通过接受一定的输入，确定一系列的输出，最终使环境达到一个新的状态。适合于嵌入式系统，涉及连续的动作与状态。

适合于嵌入式系统，用于解决简单闭环控制问题。

经典应用：空调温控，定速巡航。

#### 2.2.7 C2风格



C2架构的基本规则：

构件和连接件都有一个顶部和一个底部。

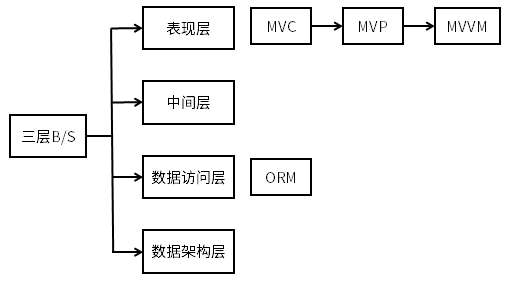
构件的顶部要连接到连接件的底部，构件的底部要连接到连接件的顶部，构件之间不允许直连

一个连接件可以和任意数目的其它构件和连接件连接

当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

### 2.3 典型架构应用

#### 2.3.1 层次架构



（1）MVC

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

J2EE体系结构中：

视图（View）：JSP

控制（Controller）：Servlet

模型（Model）：Entity Bean、Session Bean

MVP的优点：

模型与视图完全分离，我们可以修改视图而不影响模型。

可以更高效地使用模型，因为所以的交互都发生在一个地方——Presenter内部。

我们可以将一个Presener用于多个视图，而不需要改变Presenter的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁。

如果我们把逻辑放在Presenter中，那么我们就可以脱离用户接口来测试这些逻辑（单元测试）

#### 2.3.2 基于服务的架构(SOA)

（1）典型的SOA架构



（2）单个服务的内部结构



（3）SOA层次和特点



服务是标准化程度更高的构件。

服务构件粗粒度，传统构件细粒度居多（粗粒度）

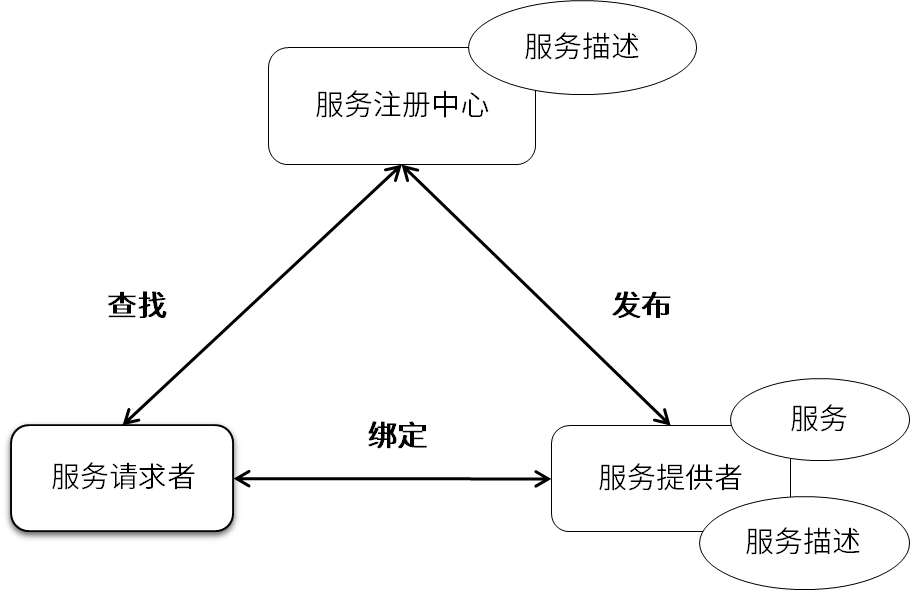
服务构件的接口是标准的，主要是WSDL接口，传统构件常以具体API形式出现（标准化结构）

服务构件的实现与语言无关，传统构件绑定某种特定语言

服务构件可以通过构件容器提供QoS的服务，传统构件完全由程序代码直接控制（松耦合）

（4）SOA的实现方式

SOA的实现方式 – Web Service



底层传输层

服务通信协议层

服务描述层

服务层

业务流程层

服务注册层

SOA的实现方式 – ESB

消息中间件、服务总线，它是面向服务架构的一种实现方式。



【服务请求者与服务提供者之间解耦】

提供位置透明性的消息路由和寻址服务

提供服务注册和命名的管理功能

支持多种的消息传递范型

支持多种可以广泛使用的传输协议

支持多种数据格式及其相互转换

提供日志和监控功能

SOA的实现方式 – 服务注册表

服务注册：应用开发者（服务提供者）向注册表公布服务的功能

服务位置：服务使用者（服务应用开发者），帮助他们查询注册服务，寻找符合自身要求的服务

服务绑定：服务使用者利用检索到的服务接口来编写代码，所编写的代码将与注册的服务绑定、调用注册的服务，以及与它们实现互动

#### 2.3.3 微服务-混合风格

（1）什么是微服务

微服务顾名思义，就是很小的服务，所以它属于面向服务架构的一种。

微服务架构是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP协议的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够被独立的部署到生产环境、类生产环境等。另外，应当尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建。

微服务在更小的基础上，其实进一步在突显其独立性。

在面向服务的架构中，提出的思想还只是从软件层面去提高独立性，而到了微服务，则进一步强调配套的数据库，甚至于部署的环境都是独立的，异或是可以独立的，这样可以进一步提高独立性。

（2）特点

小, 且专注于做⼀件事情；轻量级的通信机制；松耦合、独立部署

（3）结构





（4）微服务的优势

微服务之所以能盛行，必然是有它独特优势的。

技术异构性：在微服务架构中，每个服务都是一个相对独立的个体，每个服务都可以选择适合于自身的技术来实现。

弹性：弹性主要讲的是系统中一部分出现故障会引起多大问题。在单块系统中，一个部分出现问题，可能导致整体系统的问题。而微服务架构中，每个服务可以内置可用性的解决方案与功能降级方案，所以比单块系统强。

扩展：单块系统中，我们要做扩展，往往是整体进行扩展。而在微服务架构中，可以针对单个服务进行扩展。

简化部署：在大型单块系统中，即使修改一行代码，也需要重新部署整个应用系统。这种部署的影响很大、风险很高，因此不敢轻易的重新部署。而微服务架构中，每个服务的部署都是独立的，这样就可以更快地对特定部分的代码进行部署。

与组织结构相匹配：我们都知道，团队越大越难管理，同时团队越大也代表系统规模越大代码库越大，这样容易引起一系列的问题。且当团队是分布式的时候，问题更严重。

微服务架构就能很好的解决这个问题，微服务架构可以将架构与组织结构相匹配，避免出现过大的代码库，从而获得理想的团队大小及生产力。服务的所有权也可以在团队之间迁移，从而避免异地团队的出现。

可组合性：在微服务架构中，系统会开放很多接口供外部使用。当情况发生改变时，可以使用不同的方式构建应用，而整体化应用程序只能提供一个非常粗粒度的接口供外部使用。

对可替代性的优化：在单块系统中如果删除系统中的上百行代码，也许不知道会发生什么，引起什么样的问题，因为单块系统中关联性很强。但在微服务架构中，我们可以在需要时轻易的重写服务，或者删除不再使用的服务。

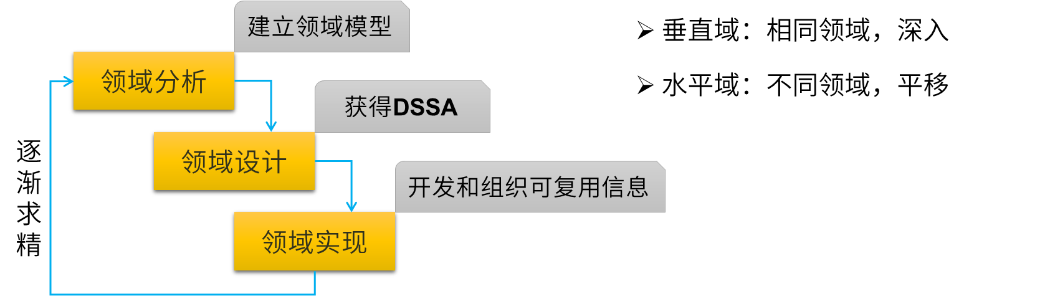
（5）微服务与SOA的对比

|  |  |
| --- | --- |
| 微服务 | SOA |
| 能拆分的就拆分 | 是整体的，服务能放一起的都放一起 |
| 纵向业务划分 | 是水平分多层 |
| 由单一组织负责 | 按层级划分不同部门的组织负责 |
| 细粒度 | 粗粒度 |
| 两句话可以解释明白 | 几百字只相当于SOA的目录 |
| 独立的子公司 | 类似大公司里面划分了一些业务单元（BU） |
| 组件小 | 存在较复杂的组件 |
| 业务逻辑存在于每一个服务中 | 业务逻辑横跨多个业务领域 |
| 使用轻量级的通信方式，如HTTP | 企业服务产总线（ESB）充当了服务之间通信的角色 |

|  |  |
| --- | --- |
| 微服务架构实现 | SOA实现 |
| 团队级，自底向上开展实施 | 企业级，自顶向下开展实施 |
| 一个系统被拆分成多个服务，粒度细 | 服务由多个子系统组成，粒度大 |
| 无集中式总线，松散的服务架构 | 企业服务总线，集中式的服务架构 |
| 集成方式简单（HTTP/REST/JSON） | 集成方式复杂（ESB/WS/SOAP） |
| 服务能独立部署 | 单块架构系统，相互依赖，部署复杂 |

### 2.4 特定领域软件架构(DSSA)（⭐⭐⭐⭐）

#### 2.4.1 基本活动



垂直域：某一个狭小领域或者说某个行业的共性抽象。

水平域：多个行业可通用的一些共性的抽象。

#### 2.4.2 领域分析机制



（1）领域专家

领域专家可能包括该领域中系统的有经验的用户、从事该领域中系统的需求分析、设计、实现以及项目管理的有经验的软件工程师等。

领域专家的主要任务包括提供关于领域中系统的需求规约和实现的知识，帮助组织规范的、一致的领域字典，帮助选择样本系统作为领域工程的依据，复审领域模型、DSSA等领域工程产品，等等。

（2）领域分析人员

领域分析人员应由具有知识工程背景的有经验的系统分析员来担任。

领域分析人员的主要任务包括控制整个领域分析过程，进行知识获取，将获取的知识组织到领域模型中，根据现有系统、标准规范等验证领域模型的准确性和一致性，维护领域模型。

领域分析人员应熟悉软件重用和领域分析方法；熟悉进行知识获取和知识表示所需的技术、语言和工具；应具有一定的该领域的经验，以便于分析领域中的问题及与领域专家进行交互；应具有较高的进行抽象、关联和类比的能力；应具有较高的与他人交互和合作的能力。

（3）领域设计人员

领域设计人员应由有经验的软件设计人员来担任。

领域设计人员的主要任务包括控制整个软件设计过程，根据领域模型和现有的系统开发出DSSA，对DSSA的准确性和一致性进行验证，建立领域模型和DSSA之间的联系。

领域设计人员应熟悉软件重用和领域设计方法；熟悉软件设计方法；应有一定的该领域的经验，以便于分析领域中的问题及与领域专家进行交互。

（4）领域实现人员

领域实现人员应由有经验的程序设计人员来担任。

领域实现人员的主要任务包括根据领域模型和DSSA，或者从头开发可重用构件，或者利用再工程的技术从现有系统中提取可重用构件，对可重用构件进行验证，建立DSSA与可重用构件间的联系。

领域实现人员应熟悉软件重用、领域实现及软件再工程技术；熟悉程序设计；具有一定的该领域的经验。

### 2.5 基于架构的软件开发方法（⭐⭐⭐⭐）

（1）基于架构的软件设计(ABSD)

ABSD能很好的支持软件重用

ABSD方法是架构驱动，即强调由业务、质量和功能需求的组合驱动架构设计。

使用ABSD方法，设计活动可以从项目总体功能框架明确就开始，这意味着需求获取和分析还没有完成（甚至远远没有完成），就开始了软件设计。

ABSD方法有三个基础。第一个基础是功能的分解。在功能分解中，ABSD方法使用已有的基于模块的内聚和耦合技术；第二个基础是通过选择架构风格来实现质量和业务需求；第三个基础是软件模板的使用。软件模板利用了一些软件系统的结构。

ABSD方法是递归的，且迭代的每一个步骤都是清晰地定义的。因此，不管设计是否完成，架构总是清晰的，这有助于降低架构设计的随意性。

（2）开发过程

架构文档化过程的主要输出结果是架构规格说明和测试架构需求的质量设计说明书这两个文档。

文档的完整性和质量是软件架构成功的关键因素。

关于文档的三大注意事项：

文档要从使用者的角度进行编写

必须分发给所有与系统有关的开发人员

且必须保证开发者手上的文档是最新的

架构复审的目的是标识潜在的风险，及早发现架构设计中的缺陷和错误

（3）架构需求过程

（4）架构设计过程

（5）架构实现过程

（6）架构演化过程

### 2.6 架构评估（⭐⭐⭐⭐⭐）

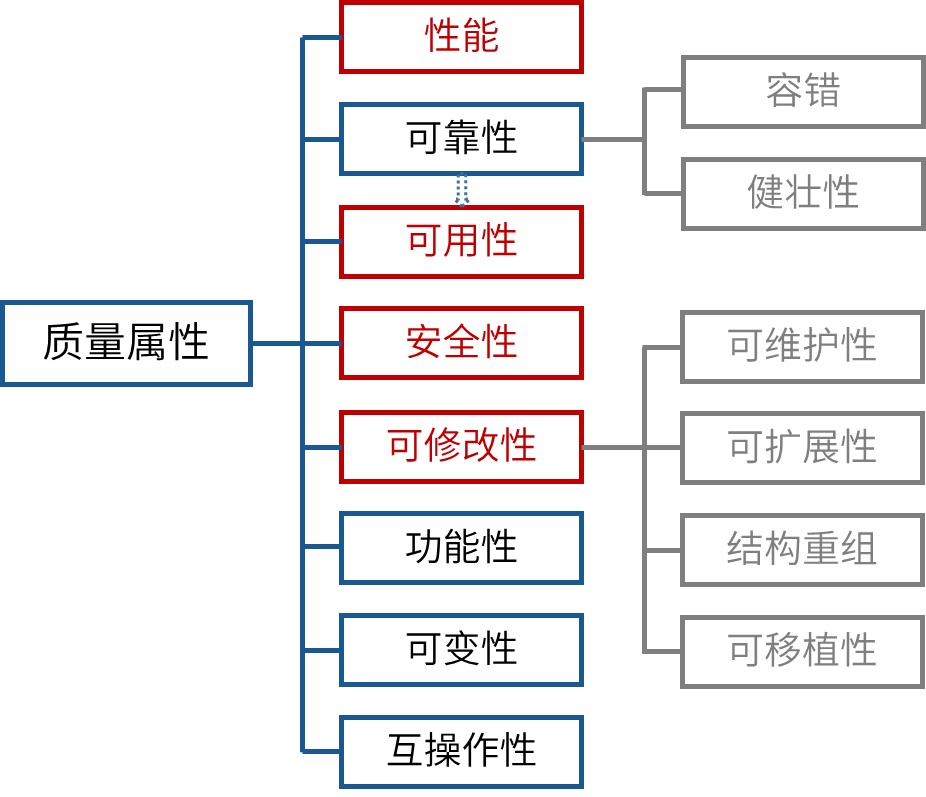
#### 2.6.1 为什么要进行架构评估？

架构是整个软件的主体结构，如果主体结构出错了，将导致整个软件开发的失败。如开发过程中需要进行大量的修整，如开发完成之后发现不合要求。这是极大的风险。

#### 2.6.2 架构评估到底评什么？

评的是这个架构好不好。所谓好，是符合需求。

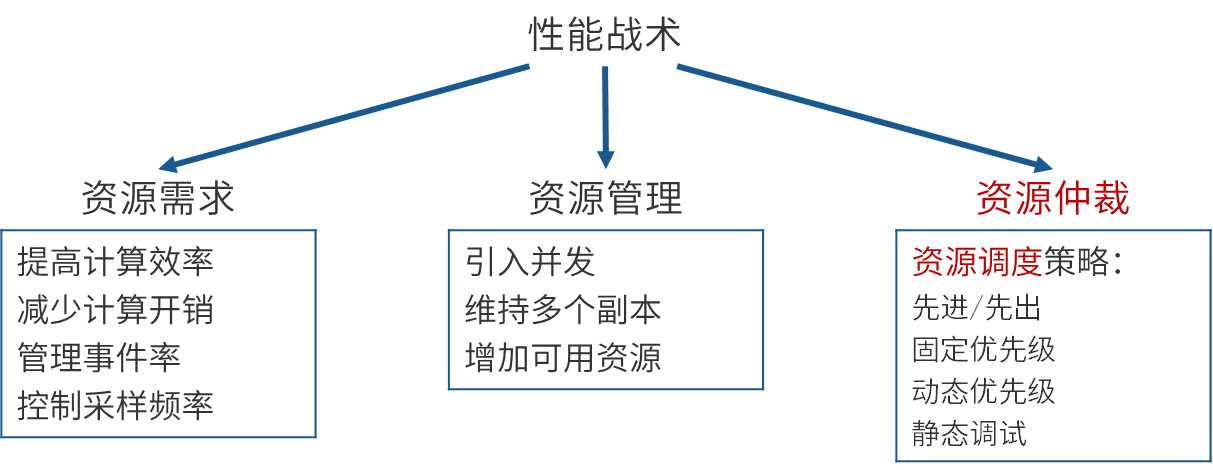
#### 2.6.3 架构设计重点关注非功能设计（质量属性）（⭐⭐⭐⭐⭐）



（1）性能

性能（performance）是指系统的响应能力，即要经过多长时间才能对某个事件做出响应，或者在某段时间内系统所能处理的事件的个数。例如：（1）同时支持1000并发；（2）响应时间小于1s；（3）显示分辨率达到4K。

代表参数：响应时间、吞吐量 设计策略：优先级队列、资源调度



（2）可靠性

可靠性（reliability）是软件系统在应用或系统错误面前，在意外或错误使用的情况下维持软件系统的功能特性的基本能力。

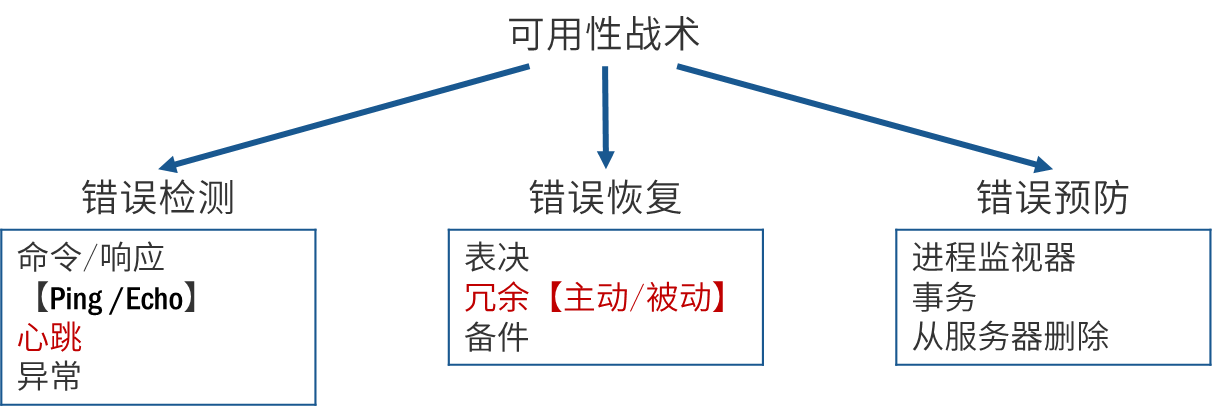
代表参数： MTTF、MTBF 设计策略：冗余、心跳线

（3）可用性

可用性（availability）是系统能够正常运行的时间比例。经常用两次故障之间的时间长度或在出现故障时系统能够恢复正常的速度来表示。例如：（1）主服务器故障，1分钟内切换至备用服务器；（2）系统故障，1小时内修复；（3）系统支持7╳24小时工作。

（可靠性与可用性意义相近，一般选择优先选择可用性。可靠性要求比可用性更高。可靠则必可用，而可用不一定可靠。可用性是可靠性的一个指标。可参照概念整理《第5章 系统可靠性分析与设计》理解。）

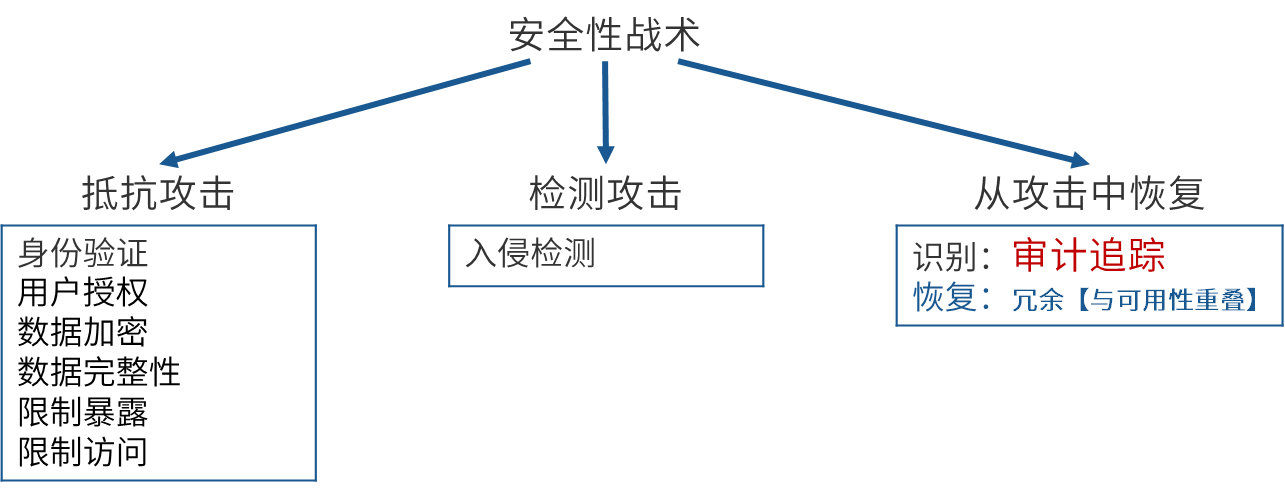
代表参数：故障间隔时间 设计策略：冗余、心跳线



（4）安全性

安全性（security）是指系统在向合法用户提供服务的同时能够阻止非授权用户使用的企图或拒绝服务的能力。安全性又可划分为机密性、完整性、不可否认性及可控性等特性。例如：（1）可抵御SQL注入攻击；（2）对计算机的操作都有完整记录；（3）用户信息数据库授权必须保证99.9%可用。

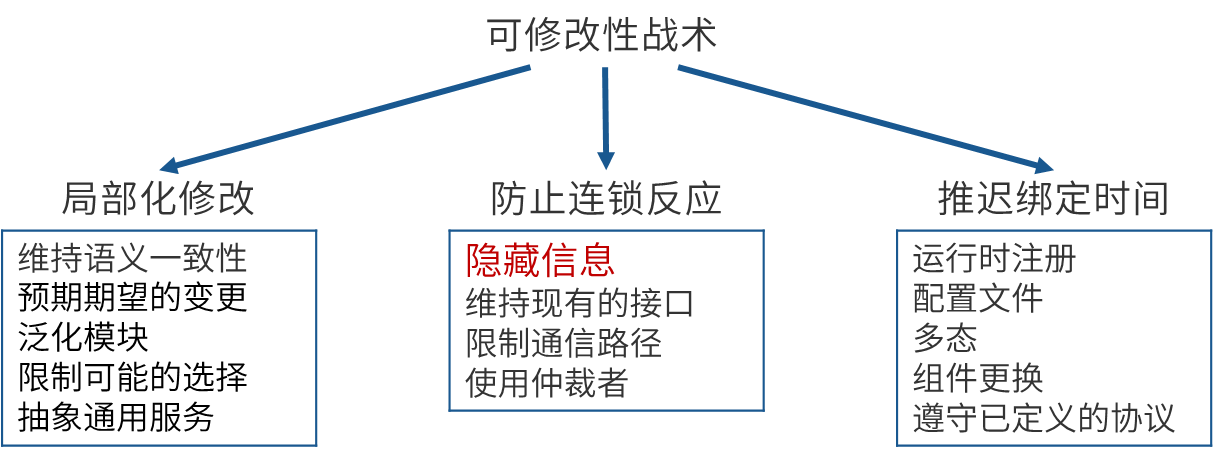
设计策略：追踪审计



（5）可修改性

可修改性（modifiability）是指能够快速地以较高的性能价格比对系统进行变更的能力。通常以某些具体的变更为基准，通过考察这些变更的代价衡量可修改性。（可扩展性与之相近）例如：（1）更改系统报表模块，必须在2人周内完成；（2）对Web界面风格进行修改，修改必须在4人月内完成。

主要策略：信息隐藏（二义性：良好的封装能够做到信息隐藏，一般归于可修改性策略；信息隐藏也能够体现在安全性当中）



（6）功能性

功能性（functionality）是系统所能完成所期望的工作的能力。一项任务的完成需要系统中许多或大多数构件的相互协作。

（7）可变性

可变性（changeability）是指体系结构经扩充或变更而成为新体系结构的能力。这种新体系结构应该符合预先定义的规则，在某些具体方面不同于原有的体系结构。当要将某个体系结构作为一系列相关产品（例如，软件产品线）的基础时，可变性是很重要的。

（8）互操作性

作为系统组成部分的软件不是独立存在的，经常与其他系统或自身环境相互作用。为了支持互操作性（interoperation），软件体系结构必须为外部可视的功能特性和数据结构提供精心设计的软件入口。程序和用其他编程语言编写的软件系统的交互作用就是互操作性的问题，这种互操作性也影响应用的软件体系结构。

（9）易用性

易用性关注的是对用户来说完成某个期望任务的容易程度和系统所提供的用户支持的种类。例如：（1）界面友好；（2）新用户学习使用系统时间不超过2小时。

（10）可测试性

软件可测试性是指通过测试揭示软件缺陷的容易程度。例如：（1）提供远程调试接口，支持远程调试。

#### 2.6.4 软件架构评估方法（⭐⭐⭐）

（1）相关概念

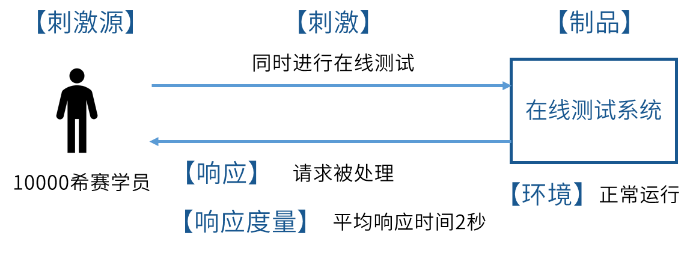
风险点：系统架构风险是指架构设计中潜在的、存在问题的架构决策所带来的隐患。

非风险点：是指不会带来隐患，一般以“XXX要求是可以实现【或接受】的”方式表达。

敏感点：指为了实现某种特定的质量属性，一个或多个构件所具有的特性。

权衡点：影响多个质量属性的特性，是多个质量属性的敏感点。

场景：用例描述功能需求，场景描述非功能需求。（一般由业务人员提出来）



性能场景示例

例如：刺激是场景中解释或描述风险承担者怎样引发与系统的交互部分。环境描述的是刺激发生时的情况。响应是指系统是如何通过体系结构对刺激作出反应的

系统目前为空闲状态，CPU利用率在5%以内，内存使用率在30%以内（此为环境），部门经理王某利用系统进行月报表统计（刺激），在5秒之后显示出了月报表（响应）。组合起来就是场景。

（2）评估方式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 评估方式 | 调查问卷或检查表 | | 场景 | 度量 |
| 调查问卷 | 检查表 |
| 通用性 | 通用 | 特定领域 | 特定系统 | 通用或特定领域 |
| 评估者对架构的了解程度 | 粗略了解 | 无限制 | 中等了解 | 精确了解 |
| 实施阶段 | 早 | 中 | 中 | 中 |
| 客观性 | 主观 | 主观 | 较主观 | 较客观 |

（3）基于场景的评估方法

过程：

确定应用领域的功能和软件架构的结构之间的映射

设计用于体现待评估质量属性的场景

分析软件架构对场景的支持程度

软件架构分析法（SAAM）

（最初用于分析架构可修改性，后扩展到其它质量属性。）



整理场景

对场景进行求精

确定场景的优先级

分配效用

形成“策略-场景-响应级别”的对应关系

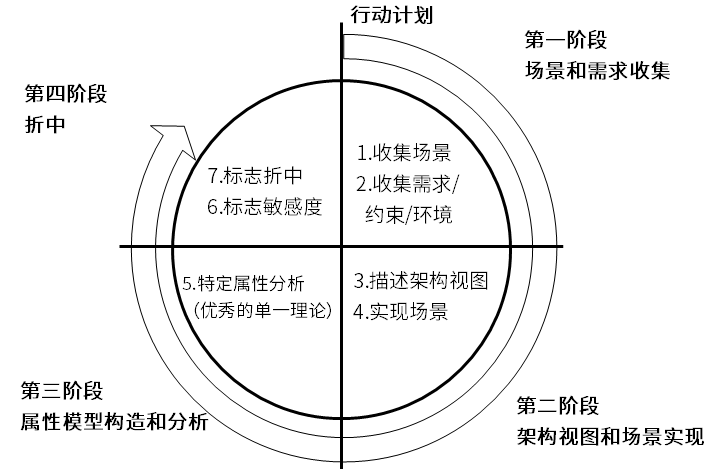
确定期望的质量属性响应级别的效用

计算各架构策略的总收益

根据受成本限制影响的投资报酬率选择架构策略

架构权衡分析法（ATAM）

（在SAAM的基础上发展起来的，主要针对性能、实用性、安全性和可修改性，在系统开发之前，对这些质量属性进行评价和折中。）



成本效益分析法（CBAM）



### 2.7 产品线

#### 2.7.1 基本概念

（产品线是多个知识领域的综合体，软件产品线会包括特定领域架构DSSA）



#### 2.7.2 特点

核心资源、产品集合，过程驱动，特定领域，技术支持，以架构为中心。

#### 2.7.3 过程模型

（1）双生命周期模型



#### 2.7.4 建立方式

|  |  |  |
| --- | --- | --- |
|  | 演化方式 | 革命方式 |
| 基于现有产品 | 基于现有产品架构设计产品线的架构，经演化现有构件，开发产品线构件 | 核心资源的开发基于现有产品集的需求和可预测的、将来需求的超集 |
| 全新产品线 | 产品线核心资源随产品新成员的需求而演化 | 开发满足所有预期产品线成员的需求的核心资源 |

将现有产品演化为产品线

用软件产品线替代现有产品集

全新软件产品线的演化

全新软件产品线的开发

#### 2.7.5 组织结构类型

设立独立的核心资源小组

不设立独立的核心资源小组

动态的组织结构

#### 2.7.6 成功实施产品线主要取决因素

对该领域具备长期和深厚的经验

一个用于构建产品的好的核心资源库

好的产品线架构

好的管理（软件资源、人员组织、过程）支持

### 2.8 WEB设计

#### 2.8.1 集群

（1）单台机器到数据库与Web服务器分离

（2）应用服务器集群

（3）主从数据库（数据库读写分离化）

主从数据库结构特点：

一般：一主多从，也可以多主多从。

主库做写操作，从库做读操作。

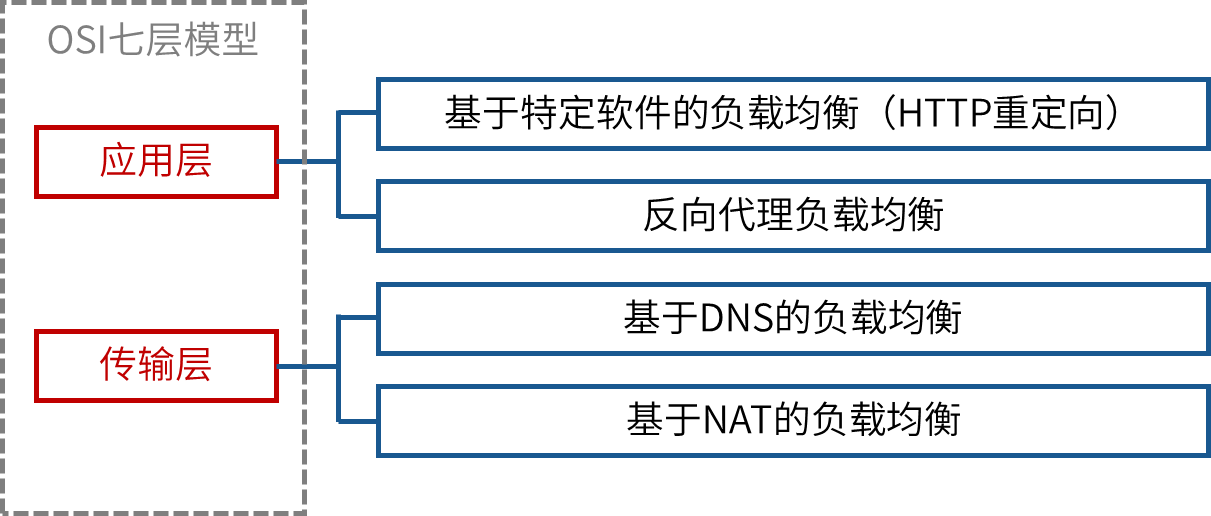
主从复制步骤：

主库（Master）更新数据完成前，将操作写binlog日志文件。

从库（Salve）打开I/O线程与主库连接，做binlog dump process，并将事件写入中继日志。

从库执行中继日志事件，保持与主库一致。

#### 2.8.2 负载均衡技术



（1）应用层负载均衡

http重定向。HTTP重定向就是应用层的请求转发。用户的请求其实已经到了HTTP重定向负载均衡服务器，服务器根据算法要求用户重定向，用户收到重定向请求后，再次请求真正的集群。

特点：实现简单，但性能较差。

反向代理服务器。在用户的请求到达反向代理服务器时（已经到达网站机房），由反向代理服务器根据算法转发到具体的服务器。常用的apache，nginx都可以充当反向代理服务器。

特点：部署简单，但代理服务器可能成为性能的瓶颈。

（2）传输层负载均衡

DNS域名解析负载均衡。DNS域名解析负载均衡就是在用户请求DNS服务器，获取域名对应的IP地址时，DNS服务器直接给出负载均衡后的服务器IP。

特点：效率比HTTP重定向高，减少维护负载均衡服务器成本。但一个应用服务器故障，不能及时通知DNS，而且DNS负载均衡的控制权在域名服务商那里，网站无法做更多的改善和更强大的管理。

基于NAT的负载均衡。基于NAT的负载均衡将一个外部IP地址映射为多个IP地址,对每次连接请求动态地转换为一个内部节点的地址。

特点：技术较为成熟，一般在网关位置，可以通过硬件实现。像四层交换机一般就采用了这种技术。

（3）硬件负载均衡：F5

（4）软件负载均衡：LVS、Nginx、HAproxy

（5）算法分类

静态算法（不考虑动态负载）：

（1）轮转算法：轮流将服务请求（任务）调度给不同的节点（即：服务器）。

（2）加权轮转算法：考虑不同节点处理能力的差异。

（3）源地址哈希散列算法：根据请求的源IP地址，作为散列键从静态分配的散列表找出对应的节点。

（4）目标地址哈希散列算法：根据请求目标IP做散列找出对应节点。

（5）随机算法：随机分配，简单，但不可控。

动态算法（考虑动态负载）

（1）最小连接数算法：新请求分配给当前活动请求数量最少的节点，每个节点处理能力相同的情况下。

（2）加权最小连接数算法：考虑节点处理能力不同，按最小连接数分配。

（3）加权百分比算法：考虑了节点的利用率、硬盘速率、进程个数等，使用利用率来表现剩余处理能力。

#### 2.8.3 Session

（1）Session共享机制



（2）有状态和无状态问题

无状态服务（stateless service）对单次请求的处理，不依赖其他请求，也就是说，处理一次请求所需的全部信息，要么都包含在这个请求里，要么可以从外部获取到（比如说数据库），服务器本身不存储任何信息。

有状态服务（stateful service）则相反，它会在自身保存一些数据，先后的请求是有关联的。

#### 2.8.4 持久化技术-ORM

ORM（Object Relational Mapping）：对象与关系数据之间的映射。

映射关系表

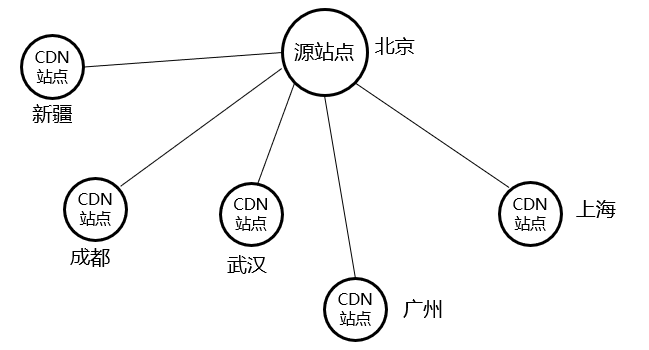
|  |  |
| --- | --- |
| 面向对象 | 关系数据库 |
| 类（class） | 数据库的表（table） |
| 对象（object） | 记录（record，行数据） |
| 对象的属性（attribute） | 字段（field） |

实现技术对比

|  |  |  |
| --- | --- | --- |
| 维度 | Hibernate | MyBatis |
| 简单对比 | 强大，复杂，间接，SQL无关 | 小巧，简单，直接，SQL相关 |
| 可移植性 | 好（不关心具体数据库） | 差（根据数据库SQL编写） |
| 复杂多表关联 | 不支持 | 支持 |

#### 2.8.5 CDN内容分发网络

CDN的全称是Content Delivery Network，即内容分发网络。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。



#### 2.8.6 缓存



（1）MemCache：Memcache是一个高性能的分布式的内存对象缓存系统，用于动态Web应用以减轻数据库负载。Memcache通过在内存里维护一个统一的巨大的hash表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。

（2）Redis：Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。

（3）Squid：Squid是一个高性能的代理缓存服务器，Squid支持FTP、gopher、HTTPS和HTTP协议。和一般的代理缓存软件不同，Squid用一个单独的、非模块化的、I/O驱动的进程来处理所有的客户端请求。

（4）Redis和Memcache对比：

|  |  |  |
| --- | --- | --- |
| 工作 | MemCache | Redis |
| 数据类型 | 简单key/value结构 | 丰富的数据结构 |
| 持久性 | 不支持 | 支持 |
| 分布式存储 | 客户端哈希分片/一致性哈希 | 多种方式，主从、Sentinel、Cluster等 |
| 多线程支持 | 支持 | 不支持（Redis6.0开始支持） |
| 内存管理 | 私有内存池/内存池 | 无 |
| 事务支持 | 不支持 | 有限支持 |
| 数据容灾 | 不支持，不能做数据恢复 | 支持，可以在灾难发生时，恢复数据 |

Redis和Memcache都是将数据存放在内存中，都是内存数据库。他们都支持key-value数据类型。同时Memcache还可用于缓存其他东西，例如图片、视频等等，Redis还支持list、set、hash等数据结构的存储。

Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。Memcache挂掉之后，数据就没了。

灾难恢复-Memcache挂掉后，数据不可恢复; Redis数据丢失后可以恢复。

在Redis中，并不是所有的数据都一直存储在内存中的。这是和Memcache相比一个最大的区别。当物理内存用完时，Redis可以将一些很久没用到的value交换到磁盘。

Redis在很多方面支持数据库的特性，可以这样说他就是一个数据库系统，而Memcache只是简单地K/V缓存。

所以在选择方面如果有持久方面的需求或对数据类型和处理有要求的应该选择Redis。

如果简单的key/value存储应该选择Memcache。

（5）Redis常见难题

缓存雪崩：大部分缓存失效 -> 数据库崩溃

解决方案：

1、缓存的高可用性

2、缓存降级

3、Redis备份

4、提前演练

缓存穿透：查询无数据返回 -> 直接查数据库

解决方案：

1、如果查询结果为空，直接设置一个默认值存放到缓存，这样第二次到缓冲中获取就有值了。设置一个不超过5分钟的过期时间，以便能正常更新缓存。

2、设置布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从避免了对底层存储系统的查询压力。

缓存预热：系统上线的后，将相关需要缓存数据直接加到缓存系统中。

解决方案：

1、直接写个缓存刷新页面，上线时手工操作。

2、数据量不大时，可以在项目启动的时候自动进行加载。

3、定时刷新缓存。

缓存更新

除Redis系统自带的缓存失效策略，常见采用以下两种：

1、定时清理过期的缓存。

2、当有用户请求过来时，再判断这个请求所用到的缓存是否过期，过期的话就去底层系统得到新数据并更新缓存。

缓存降级

降级的目的是保证核心服务可用，即使是有损的，而且有些服务是无法降级的（如电商的购物流程等）

在进行降级之前要对系统进行梳理，从而梳理出哪些必须保护，哪些可降级。

（6）缓存与数据库的协作

数据库读取：

根据key从缓存读取；若缓存中没有，则根据key在数据库中查找；读取到“值”之后，更新缓存。

数据库写入

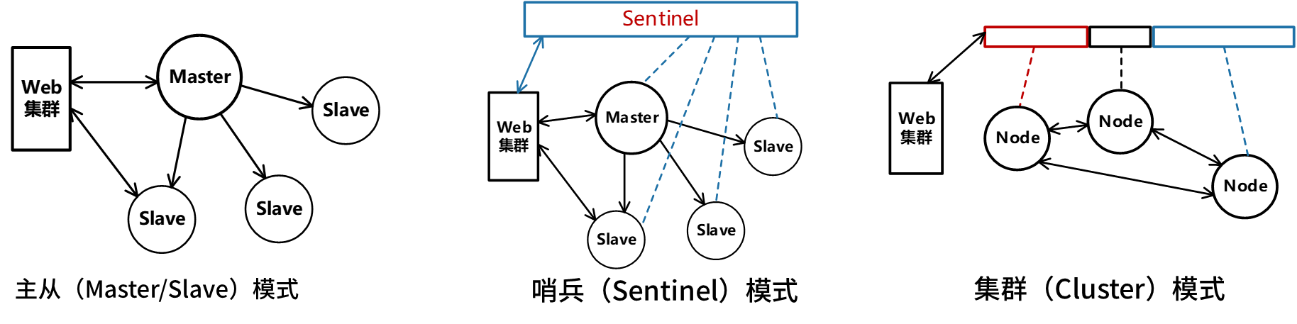
根据key值写数据库；根据key更新缓存。

（7）Redis集群切片方式



|  |  |
| --- | --- |
| 集群切片方式 | 核心特点 |
| 客户端分片 | 在客户端通过key的hash值对应到不同的服务器。 |
| 中间件实现分片 | 在应用软件和Redis中间，例如：Twemproxy、Codis等，由中间件实现服务到后台Redis节点的路由分派。 |
| 客户端服务端协作分片 | RedisCluster模式，客户端可采用一致性哈希，服务端提供错误节点的重定向服务slot上。不同的slot对应到不同服务器。 |

（8）Redis分布存储方案



|  |  |
| --- | --- |
| 分布式存储方案 | 核心特点 |
| 主从（Master/Slave）模式 | 一主多从，故障时手动切换。 |
| 哨兵（Sentinel）模式 | 有哨兵的一主多从，主节点故障自动选择新的主节点。 |
| 集群（Cluster）模式 | 分节点对等集群，分slots，不同slots的信息存储到不同节点。 |

（9）Redis分片方案

|  |  |  |
| --- | --- | --- |
| 分片方案 | 分片方式 | 说明 |
| 范围分片 | 按数据范围值来做分片 | 例：按用户编号分片，0-999999映射到实例A；1000000-1999999映射到实例B。 |
| 哈希分片 | 通过对key进行hash运算分片 | 可以把数据分配到不同实例，这类似于取余操作，余数相同的，放在一个实例上。 |
| 一致性哈希分片 | 哈希分片的改进 | 可以有效解决重新分配节点带来的无法命中问题。 |

（10）Redis数据类型

|  |  |  |
| --- | --- | --- |
| 类型 | 特点 | 示例 |
| String（字符串） | 存储二进制，任何类型数据，最大512MB | 缓存，计数，共享Session |
| Hash（字典） | 无序字典，数组+链表，适合存对象。  Key对应一个HashMap。针对一组数据 | 存储、读取、修改用户属性 |
| List（列表） | 双向链表，有序，增删快查询慢 | 消息队列，文章列表  记录前N个最新登陆的用户ID列表 |
| Set（集合） | 键值对无序，唯一  增删查复杂度均为O（1）,支持交/并/差集操作 | 独立IP，共同好爱，标签 |
| Sorted Set（有序集合） | 键值对有序，唯一，自带按权重排序效果 | 排行榜 |

（11）Redis持久化

RDB：传统数据库中快照的思想。指定时间间隔将数据进行快照存储。

AOF：传统数据库中日志的思想，把每条改变数据集的命令追加到AOF文件末尾，这样出问题了，可以重新执行AOF文件中的命令来重建数据集。

|  |  |  |
| --- | --- | --- |
| 对比维度 | RDB持久化 | AOF持久化 |
| 备份量 | 重量级的全量备份，保存整个数据库 | 轻量级增量备份，一次只保存一个修改命令 |
| 保存间隔时间 | 保存间隔时间长 | 保存间隔时间知，默认1秒 |
| 还原速度 | 数据还原速度快 | 数据还原速度慢 |
| 阻塞情况 | save会阻塞，但bgsave或者自动不会阻塞 | 无论是平时还是AOF重写，都不会阻塞 |
| 数据体积 | 同等数据体积 : 小 | 同等数据体积 : 大 |
| 安全性 | 数据安全性 : 低，容易丢数据 | 数据安全性 : 高，根据策略决定 |

（12）淘汰机制

|  |  |  |
| --- | --- | --- |
| 淘汰作用范围 | 机制名 | 策略 |
| 不淘汰 | noeviction | 禁止驱逐数据，内存不足以容纳新入数据时，新写入操作就会报错。系统默认的一种淘汰策略。 |
| 设置了过期时间的键空间 | volatile-random | 随机移除某个key |
| volatile-lru | 优先移除最近未使用的key |
| volatile-ttl | ttl值小的key优先移除 |
| 全键空间 | allkeys-random | 随机移除某个key |
| allkeys-lru | 优先移除最近未使用的key |

#### 2.8.7REST

（1）概念

REST（Representational State Transfer，表述性状态转移）是一种只使用HTTP和XML进行基于Web通信的技术，可以降低开发的复杂性，提高系统的可伸缩性。

（2）REST的5个原则

网络上的所有事物都被抽象为资源。

每个资源对应一个唯一的资源标识。

通过通用的连接件接口对资源进行操作。

对资源的各种操作不会改变资源标识。

所有的操作都是无状态的。

## 3 章节问答

（1）面向对象风格、面向对象的开发和独立构件风格如何区分？一个软件开发中，可以同时用到多种架构风格吗？

答：

面向对象的设计面向对象开发方法的一环，属于一种开发方法，比如结构化开发方法，面向对象开发方法。

架构风格中：

调用返回风格中（面向对象风格中）的构件就是对象，对象是通过函数和过程的调用来交互的。

独立构件风格中（进程通信）的构件就是进程或事件，而进程的通信就是通过消息进行通信的，这种消息可以是共享信息。

另外，在一个软件开发种，是可以同时用到多种架构风格的。

（2）拟机架构风格解释器和规则系统之间的差异点在哪里？

答：

他们都是虚拟机风格。

解释器风格是一个用来执行其它程序的程序。解释器可针对不同的硬件平台实现一个虚拟机，将高抽象层次的程序翻译为低抽象层次所能理解的指令，以消除在程序语言与硬件之间存在的语义差异。

基于规则的系统是以规则为中心的虚拟机系统。而规则系统是适合于专家系统和DSS的。

（3）由构件引起失配包括系统对构件基础设施、构件控制模型、构件数据模型的假设存在冲突。这些是什么意思，能否简单举例？

答：

失配是指在软件复用的过程中，由于待复用构件对最终系统的体系结构和环境的假设与实际状况不同而导致的冲突。

比如由构件引起的失配，包括由于系统对构件基础设施、构件控制模型和构件数据模型的假设存在冲突引起的失配。比如说：而系统对构件基础设施就包括了异构构件互操作的标准和通信平台（比如omg标准、com标准、ejb标准)或者比如说：基于构件的软件过程中存在大量的并行活动,提出的基于Petri网的基于构件的软件控制模型,易于描述这些并行活动,并且能有效地控制基于构件的软件过程的实施,使过程中的活动有条不紊地向前推进。

考试的时候，一般并不会考到具体方法的细节，了解一下总概情况即可。比如“由构件引起的失配，包括由于系统对构件基础设施、构件控制模型和构件数据模型的假设存在冲突引起的失配”,重点是知道这是由构件适配引起的。

（4）质量属性场景，质量属性，质量场景三者是什么关系？

答：

质量属性场景实际上就是质量属性和质量场景的统称。那么质量属性和质量场景就好区分了。

比如这里的质量属性场景，指的是质量场景而不是质量属性。我们在使用用例来来捕获功能需求的同时，通过定义特定场景来捕获质量需求，并称这些场景为质量场景

# 第四章 信息安全分析与设计

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 安全基础技术  （⭐⭐⭐） | 对称加密与非对称加密技术  （⭐） |
| 2 | 信息摘要与数字签名  （⭐） |
| 3 | 数字证书  （⭐） |
| 4 | 其他 |
| 1 | 安全网络协议（⭐） | 网络安全协议 |
| 1 | 网络攻击（⭐⭐） | 网络攻击 |
| 1 | 网络安全防范体系（⭐） | 网络安全防范体系 |
| 1 | 其他（⭐） | 其他 |

## 2 考点精讲

### 2.1 安全基础技术（⭐⭐）

#### 2.1.1 对称加密

概念

对称加密（又称为私人秘钥加密/共享秘钥加密）：加密与解密使用同一秘钥。

特点

加密强度不高，但效率高

密钥分发困难。

（大量明文为了保证加密效率一般使用对称加密）

常见对称密钥加密算法

DES：替换+移位、56位密钥、64位数据块、速度快、密钥易产生

3DES(三重DES)：两个56位的密钥K1、K2

加密：K1加密->K2解密->K1加密

解密：K1解密->K2加密->K1解密 RC-5

IDEA：128位密钥、64位数据块、比DES的加密性好、对计算机功能要求相对低，PGP。

RC-5算法：RSA数据安全公司的很多产品都使用了RC-5。

AES算法：高级加密标准，又称Rijndael加密法，是美国政府采用的一种区块加密标准。

#### 2.1.2 非对称加密

概念

非对称加密（又称为公开密钥加密）：密钥必须成对使用（公钥加密，相应的私钥解密）。

特点

加密速度慢，但强度高。

常见非对称密钥加密算法

RSA：2048位（或1024位）密钥、计算量极大、难破解

ECC-椭圆曲线算法

Elgamal：安全性依赖于计算有限域上离散对数这一难题。

#### 2.1.3 信息摘要与数字签名

（1）数字签名的过程如下图所示（发送者使用自己的私钥对摘要签名，接收者利用发送者的公钥对接收到的摘要进行验证）：



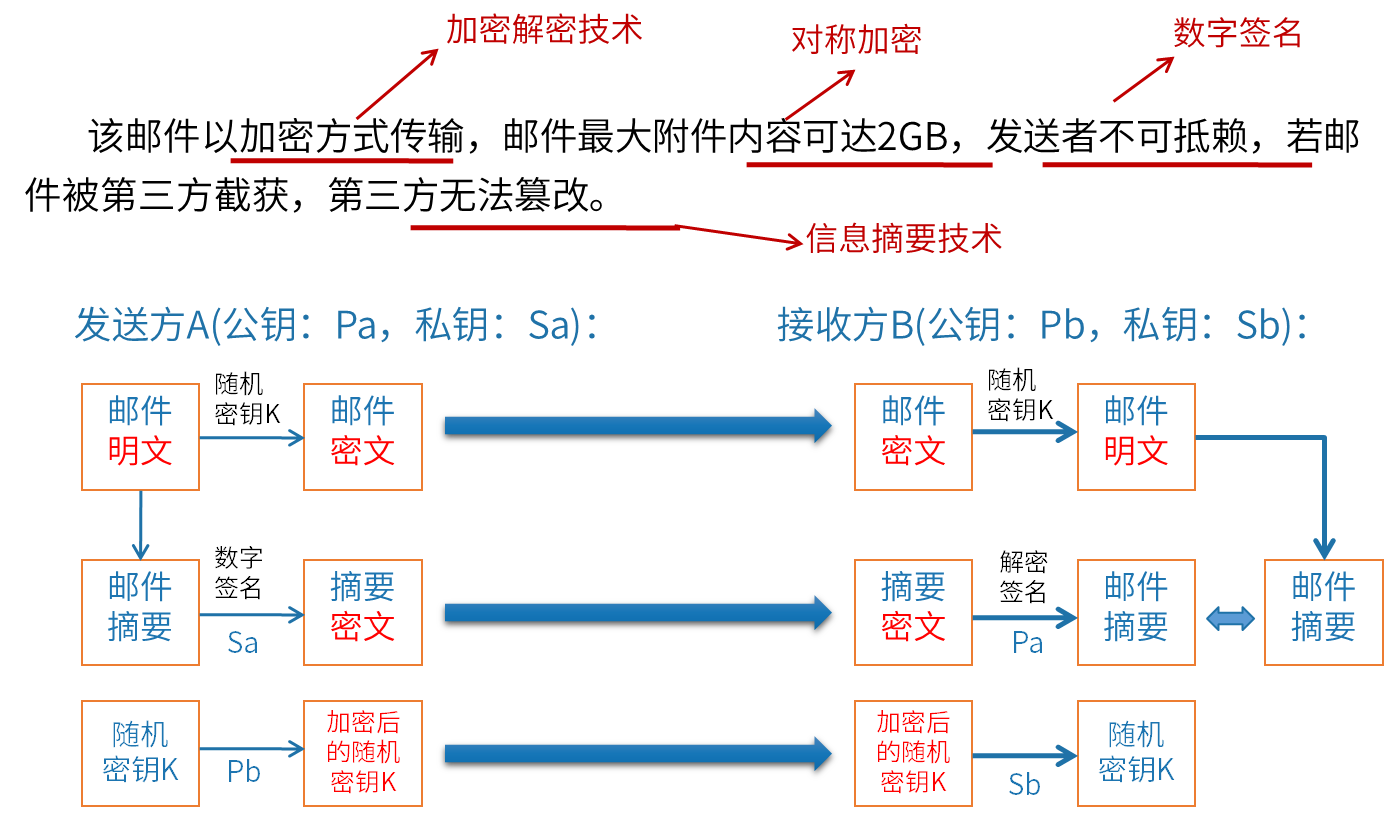
（2）常见的摘要算法：MD5(128位)，SHA(160位)。

#### 2.1.4 数字信封

发送方将原文用对称密钥加密传输，而将对称密钥用接收方公钥加密发送给对方。

接收方收到电子信封，用自己的私钥解密信封，取出对称密钥解密得原文。

#### 2.1.5 设计实例



#### 2.1.6 PGP

PGP可用于电子邮件，也可以用于文件存储。采用了杂合算法，包括IDEA、RSA、MD5、ZIP数据压缩算法。

PGP承认两种不同的证书格式：PGP证书和X.509证书。

PGP证书包含PGP版本号、证书持有者的公钥、证书持有者的信息、证书拥有者的数字签名、证书的有效期、密钥首选的对称加密算法。

X.509证书包含证书版本、证书的序列号、签名算法标识、证书有效期、以下数据：证书发行商名字、证书主体名、主体公钥信息、发布者的数字签名。

### 2.2 网络安全

#### 2.2.1 安全协议（⭐⭐）



HTTPS协议是HTTP协议与SSL协议的结合，默认端口号443。

PGP：针对邮件的混合加密系统。

SSL：工作在传输层至应用层。

TLS：传输层安全协议。

SET：电子商务。

IPSEC：对IP包加密。

#### 2.2.2 网络攻击（⭐⭐）

分类

被动攻击：收集信息为主，破坏保密性。

|  |  |  |
| --- | --- | --- |
| 攻击类型 | 攻击名称 | 描述 |
| 被动攻击 | 窃听（网络监听） | 用各种可能的合法或非法的手段窃取系统中的信息资源和敏感信息。 |
| 业务流分析 | 通过对系统进行长期监听，利用统计分析方法对诸如通信频度、通信的信息流向、通信总量的变化等参数进行研究，从而发现有价值的信息和规律。 |
| 非法登录 | 有些资料将这种方式归为被动攻击方式。 |

主动攻击：主动攻击的类别主要有：中断（破坏可用性），篡改（破坏完整性），伪造（破坏真实性）

|  |  |  |
| --- | --- | --- |
| 攻击类型 | 攻击名称 | 描述 |
| 主动攻击 | 假冒身份 | 通过欺骗通信系统（或用户）达到非法用户冒充成为合法用户，或者特权小的用户冒充成为特权大的用户的目的。黑客大多是采用假冒进行攻击。 |
| 抵赖 | 这是一种来自用户的攻击，比如：否认自己曾经发布过的某条消息、伪造一份对方来信等。 |
| 旁路控制  【旁路攻击】 | 密码学中是指绕过对加密算法的繁琐分析，利用密码算法的硬件实现的运算中泄露的信息。如执行时间、功耗、电磁辐射等，结合统计理论快速的破解密码系统。 |
| 重放攻击 | 所截获的某次合法的通信数据拷贝，出于非法的目的而被重新发送。  加时间戳能识别并应对重放攻击。 |
| 拒绝服务（DOS） | 对信息或其它资源的合法访问被无条件地阻止。 |

常见的攻击行为

拒绝服务：攻击者利用众多傀儡主机向服务器发送服务请求，导致服务器资源被耗尽，无法提供正常的服务，向其他访问者发送拒绝服务应答。

重放攻击：攻击者抓取向服务器发送的有效数据包，并利用此数据包不断地向服务器发送，导致服务器一直应答此数据包，从而崩溃。

业务流分析：通过长期监听被攻击者的数据流，从而分析出相关业务流，可以依此了解被攻击者的一些倾向，常见的广告推送就是建立在业务流分析基础上的。

常见的防御手段（可以结合使用）

防火墙技术：主要了解它的机制是防外不防内，对于DMZ非军事区主要放置应用服务器（如邮件服务器，WEB服务器）。

漏洞扫描：入侵者可以利用系统漏洞侵入系统，系统管理员可以通过漏洞扫描技术，及时了解系统存在的安全问题，并采取相应措施来提高系统的安全性。

入侵检测IDS：基于数据源的分类-审计功能、记录安全性日志。基于检测方法-异常行为检测。

### 2.3 安全保护等级

|  |  |  |  |
| --- | --- | --- | --- |
|  | 公民、法人和其它组织权益 | 社会秩序和公共利益 | 国家安全 |
| 用户自主保护级 | 损害 |  |  |
| 系统审计保护级 | 严重损害 | 损害 |  |
| 安全标记保护级 |  | 严重损害 | 损害 |
| 结构化保护级 |  |  | 严重损害 |
| 访问验证保护级 |  |  | 特别严重损害 |

用户自主保护级：适用于普通内联网用户

系统审计保护级：适用于通过内联网或国际网进行商务活动，需要保密的非重要单位

安全标记保护级：适用于地方各级国家机关、金融机构、邮电通信、能源与水源供给部门、交通运输、大型工商与信息技术企业、重点工程建设等单位

结构化保护级：适用于中央级国家机关、广播电视部门、重要物资储备单位、社会应急服务部门、尖端科技企业集团、国家重点科研机构和国防建设等部门

访问验证保护级：适用于国防关键部门和依法需要对计算机信息系统实施特殊隔离的单位

### 2.4 安全防范体系

（1）物理环境的安全性。包括通信线路、物理设备和机房的安全等。物理层的安全主要体现在通信线路的可靠性（线路备份、网管软件和传输介质）、软硬件设备的安全性（替换设备、拆卸设备、增加设备）、设备的备份、防灾害能力、防干扰能力、设备的运行环境（温度、湿度、烟尘）和不间断电源保障等。

（2）操作系统的安全性。主要表现在三个方面，一是操作系统本身的缺陷带来的不安全因素，主要包括身份认证、访问控制和系统漏洞等；二是对操作系统的安全配置问题；三是病毒对操作系统的威胁。

（3）网络的安全性。网络层的安全问题主要体现在计算机网络方面的安全性，包括网络层身份认证、网络资源的访问控制、数据传输的保密与完整性、远程接入的安全、域名系统的安全、路由系统的安全、入侵检测的手段和网络设施防病毒等。

（4）应用的安全性。由提供服务所采用的应用软件和数据的安全性产生，包括Web服务、电子邮件系统和DNS等。此外，还包括病毒对系统的威胁。

（5）管理的安全性。包括安全技术和设备的管理、安全管理制度、部门与人员的组织规则等。管理的制度化极大程度地影响着整个计算机网络的安全，严格的安全管理制度、明确的部门安全职责划分与合理的人员角色配置，都可以在很大程度上降低其他层次的安全漏洞。

## 3 章节问答

（1）消息摘要的作用是什么？对摘要加密的作用是什么？

答：

消息摘要的作用是防篡改，因为摘要是单向不可逆的，一旦篡改就得不到原来的摘要了。

对摘要进行加密，其实就是我们常说的数字签名的过程，数字签名是利用发送方的私钥对摘要进行加密的过程，我们一般称之为数字签名，这个过程是为了标识发送者身份，保证发送者身份不可抵赖。

（2）提高安全性的策略有哪些？

答：

提高安全性的手段包括：身份认证、限制访问、检测攻击、维护完整性等。

# 第五章 系统可靠性分析与设计

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 基本概念 | 基本概念 |
| 1 | 可靠性分析（⭐） | 可靠性指标 |
| 2 | 可靠性计算 |
| 1 | 可靠性设计（⭐⭐） | 可靠性设计 |

## 2 考点精讲

### 2.1 相关基本概念

可靠性：系统可靠性是系统在规定的时间内及规定的环境条件下，完成规定功能的能力，也就是系统无故障运行的概率。

可用性:系统可用性是指在某个给定时间点上系统能够按照需求执行的概率。

软件可靠性 ≠ 硬件可靠性

### 2.2 系统可靠性分析

#### 2.2.1 可靠性指标



平均无故障时间 → (MTTF) MTTF=1/λ，λ为失效率

平均故障修复时间 → (MTTR) MTTR=1/μ，μ为修复率

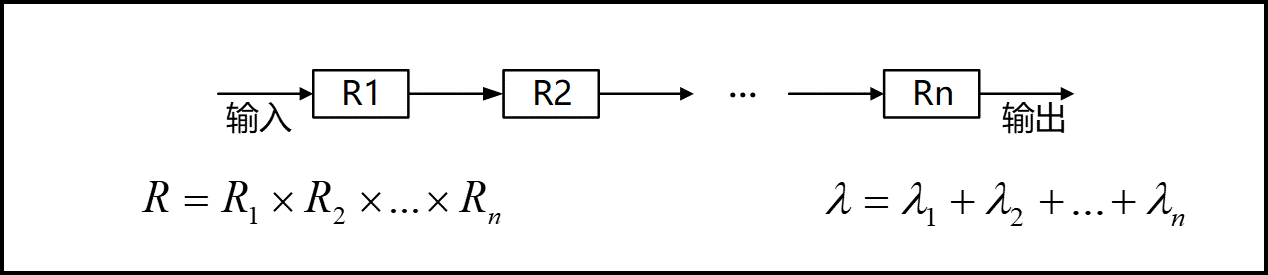
平均故障间隔时间 → (MTBF) MTBF = MTTR + MTTF

系统可用性 → MTTF/(MTTR+MTTF)×100%

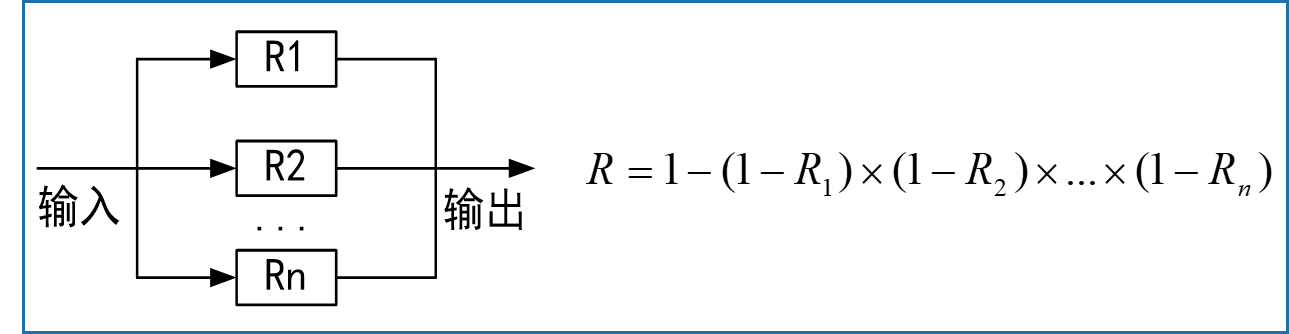
在实际应用中，一般MTTR很小，所以通常认为MTBF≈MTTF。

#### 2.2.2 可靠性计算

串联系统



并联系统

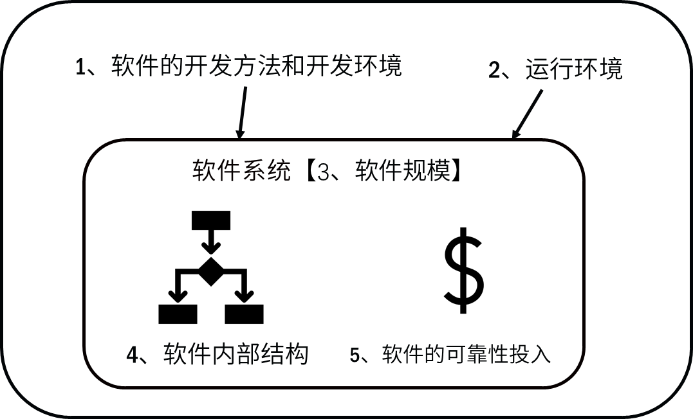


N模混联系统

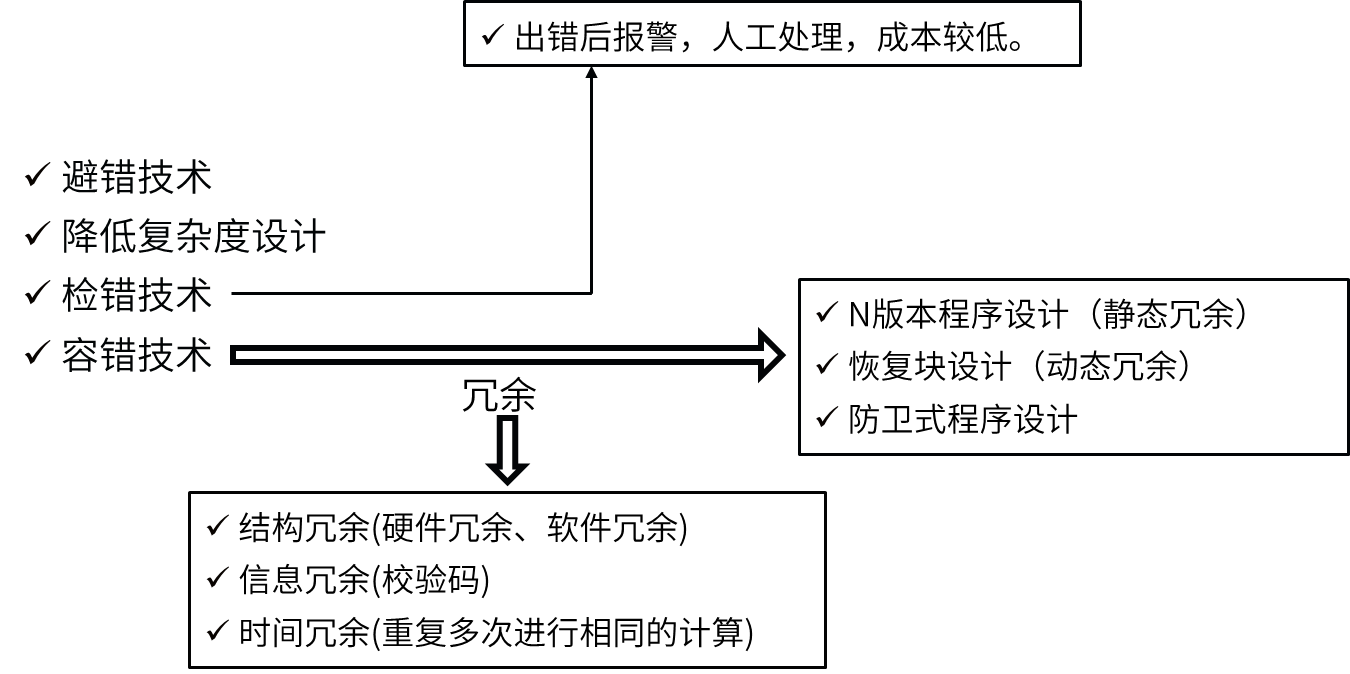
先将整个系统划分为多个部分串联R1、R2…等，再计算R1、R2内部的并联可靠性，带入原公式。

### 2.3 可靠性设计

（1）影响可靠性的因素



（2）可靠性设计技术



#### 2.3.1 N版本程序设计



与通常软件开发过程不同的是，N版本程序设计增加了三个新的阶段：相异成份规范评审、相异性确认、背对背测试

N版本程序的同步、N版本程序之间的通信、表决算法（全等表决、非精确表决、Cosmetie表决）、一致比较问题、数据相异性

#### 2.3.2 恢复块方法



设计时应保证实现主块和后备块之间的独立性，避免相关错误的产生，使主块和备份块之间的共性错误降到最低程度。

必须保证验证测试程序的正确性。

（1）恢复方法分类

前向恢复：使当前的计算继续下去，把系统恢复成连贯的正确状态，弥补当前状态的不连贯情况

后向恢复：系统恢复到前一个正确状态，继续执行

（2）恢复块方法与N版本程序设计对比

|  |  |  |
| --- | --- | --- |
|  | 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 |

#### 2.3.3 防卫式程序设计

对于程序中存在的错误和不一致性，通过在程序中包含错误检查代码和错误恢复代码，使得一旦错误发生，程序能撤消错误状态，恢复到一个已知的正确状态中去

实现策略：错误检测、破坏估计、错误恢复

#### 2.3.4 双机容错



双机热备模式（主系统、备用系统）

双机互备模式（同时提供不同的服务，心不跳则接管）

双机双工模式（同时提供相同的服务，集群的一种）

#### 2.3.5 集群技术

（1）典型的集群系统具有如下特征：

可伸缩性。采用集群技术，当用户需要扩展系统计算能力时，系统能在不降低服务质量的前提下进行扩展。一般只需购买新的计算机，将其加入到集群系统中即可，而不需要将现有的计算机更换为高性能的服务器。

高可用性。集群系统的可靠性与单机系统相比较高，在提高了系统的可靠性的同时，可以大大减小由于故障造成的停运。集群系统在部分硬件和软件发生故障时，整个系统仍高度可用，可以将系统停运的时间减到最小。

可管理性。集群系统能够管理大规模和物理分散的节点。

高性价比。集群系统能够以最少的投资获得最大的性能。在达到同样性能的条件下，采用计算机集群比采用同等运算能力的大型计算机具有更高的性价比。

高透明性。集群系统对用户是透明的，在用户看来，集群是一个系统，而非多个计算机系统。当集群系统的节点发生变化时，上层应用无需修改或尽可能少修改。

（2）集群系统与高性能主机系统对比

可用性与可靠性：采用高性能的主机系统很难实现系统的冗余机制，当发生故障后会导致整个系统失去响应能力，从而无法保证系统的高可用性。

可扩展性：负载均衡集群可扩展性强于高性能的主机。高性能主机向上扩展提升系统的可扩展性，具体的实现方式包括硬件扩容（增加CPU数量、内存容量、磁盘数量）和硬件升级（更换为高端主机或高速磁盘等）成本高，可扩展性差；集群水平扩展，通过增加主机数量、负载均衡技术进行扩展，成本较低，可扩展性较好。

成本：仅从单次实施成本来看，也许高性能主机与集群相当，但从长久来看，负载均衡系统在性能不能达到业务要求时，只需要直接增加集群机器数量，能以极低的成本进行扩展，而高性能主机则不行。

应用角度：Internet中的Web应用绝大多数是简单任务、高强度的并发处理，而主机系统只是对于复杂单一任务和有限的并发处理具有高性能，不能同时处理大量的并发请求。从之方面看，集群也占优势。

## 3 章节问答

（1）主动冗余是什么？

答：

主动冗余是指，处理结点出故障之后，结点本身或控制器会协调没有出故障的结点提供处理服务。

被动冗余，则是靠请求者自己完成这个工作，请求者如果发现一台服务器不能正常提供服务，转而找另一台服务器。

（2）负载均衡是否算是提高可靠性的策略？提高可用性的策略是否也算是提高可靠性设计的策略？

答：

提高可靠性最主要的技术是检错技术、容错设计和降低复杂度设计。

可靠性和可用性这两个质量属性是相互促进的作用。提高可用性的同时，间接在提高系统的可靠性，所以，他们的设计策略在一定程度上是有相互包含的。

负载均衡在某些角度来看算是提高可靠性的策略。比如使用带有负载平衡的多个服务器组件，取代单一的组件，可以通过冗余提高可靠性。

# 第六章 项目管理

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 立项管理 | 盈亏平衡管理 |
| 1 | 范围管理（⭐⭐⭐⭐） | 范围管理 |
| 1 | 时间管理（⭐⭐⭐） | Gantt图PERT图 |
| 2 | 其他 |

## 2 考点精讲

### 2.1 立项管理—盈亏平衡管理

销售额=固定成本+可变成本+税费+利润 【正常情况下】

销售额=固定成本+可变成本+税费 【盈亏平衡时】

### 2.2 范围管理（⭐⭐⭐）

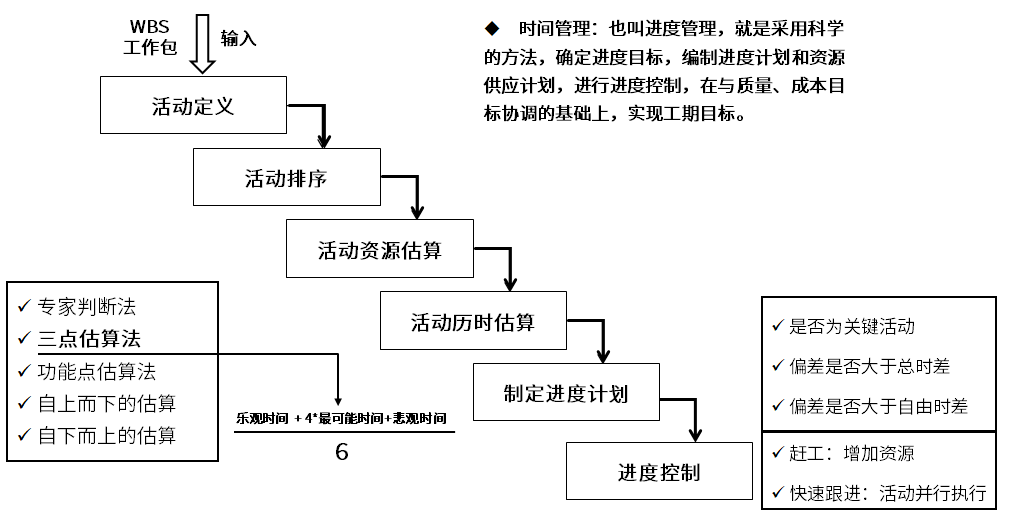
范围管理：确定项目的边界，即哪些工作是项目应该做的，哪些工作不应该包括在项目中。



### 2.3 时间管理（进度管理）（⭐⭐⭐）

时间管理：也叫进度管理，就是采用科学的方法，确定进度目标，编制进度计划和资源供应计划，进行进度控制，在与质量、成本目标协调的基础上，实现工期目标。

#### 2.3.1 过程



#### 2.3.2 历时估算方法

专家判断法

三点估算法

功能点估算法

自上而下的估算

自下而上的估算

#### 2.3.3 进度控制

可能出现的问题

是否为关键活动

偏差是否大于总时差

偏差是否大于自由时差

采取的手段

赶工：增加资源 —加班

快速跟进：活动并行执行

#### 2.3.4 进度网络图-关键路径法（PERT）

关键路径法是在制订进度计划时使用的一种进度网络分析技术。关键路线法沿着项目进度网络路线进行正向与反向分析，从而计算出所有计划活动理论上的最早开始与完成日期、最迟开始与完成日期，不考虑任何资源限制。

如下图：单代号网络图



ES：最早开始时间 EF：最早完成时间

LS：最迟开始时间 LF：最迟完成时间

总时差（松弛时间）

在不延误总工期的前提下，该活动的机动时间。活动的总时差等于该活动最迟完成时间与最早完成时间之差，或该活动最迟开始时间与最早开始时间之差

自由时差

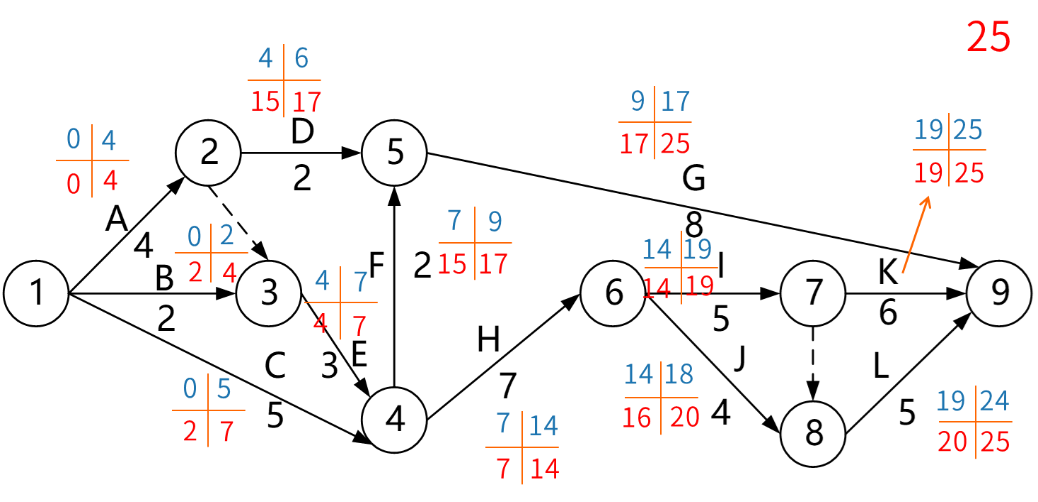
在不影响紧后活动的最早开始时间前提下，该活动的机动时间。

对于有紧后活动的活动，其自由时差等于所有紧后活动最早开始时间减本活动最早完成时间所得之差的最小值

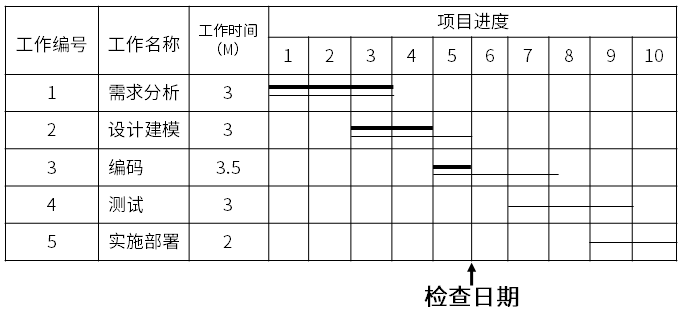
对于没有紧后活动的活动，也就是以网络计划终点节点为完成节点的活动，其自由时差等于计划工期与本活动最早完成时间之差

对于网络计划中以终点节点为完成节点的活动，其自由时差与总时差相等。此外，由于活动的自由时差是其总时差的构成部分，所以，当活动的总时差为零时，其自由时差必然为零，可不必进行专门计算

双代号网络图



#### 2.3.5 Gantt图



优点：甘特图直观、简单、容易制作，便于理解，能很清晰地标识出直到每一项任务的起始与结束时间，一般适用比较简单的小型项目，可用于WBS的任何层次、进度控制、资源优化、编制资源和费用计划。

缺点：不能系统地表达一个项目所包含的各项工作之间的复杂关系，难以进行定量的计算和分析，以及计划的优化等。

与PERT图对比

PERT（项目评估与评审技术）图是一种图形化的网络模型，描述一个项目中任务和任务之间的关系，每个节点表示一个任务，通常包括任务编号、名称、开始和结束时间、持续时间和松弛时间。

Gantt图是一种简单的水平条形图，它以一个日历为基准描述项目任务，横坐标表示时间，纵坐标表示任务，图中的水平线段表示对一个任务的进度安排，线段的起点和终点对应在横坐标上的时间分别表示该任务的开始时间和结束时间，线段的长度表示完成该任务所需的时间。

PERT图主要描述不同任务之间的依赖关系；Gantt图主要描述不同任务之间的重叠关系。

## 3 章节问答

（1）产品范围在立项和范围管理中有什么清晰度的区别吗？

答：

企业在项目立项之前，开始会由第三方或本公司做一个项目的方案建议书，其中就包括了初步的产品范围，仅仅是用在项目招投标的建议书中，这个范围肯定是比较粗的。（具体到什么程度，这个没有标准，大概就是功能结构图这种程度，然后就是对各个模块大致的描述）

而公司在中标后，进入项目建设的阶段，这个时候是需要经过详细的需求分析后，编写需求功能规格说明书（SRS），可能会跟合同中有些许差别，那么对于项目管理的人员会将合同和需求分析后的差异体现处理，因为后面验收就是以SRS和项目经理编写的差异内容进行验收。

# 第七章 计算机组成与体系结构

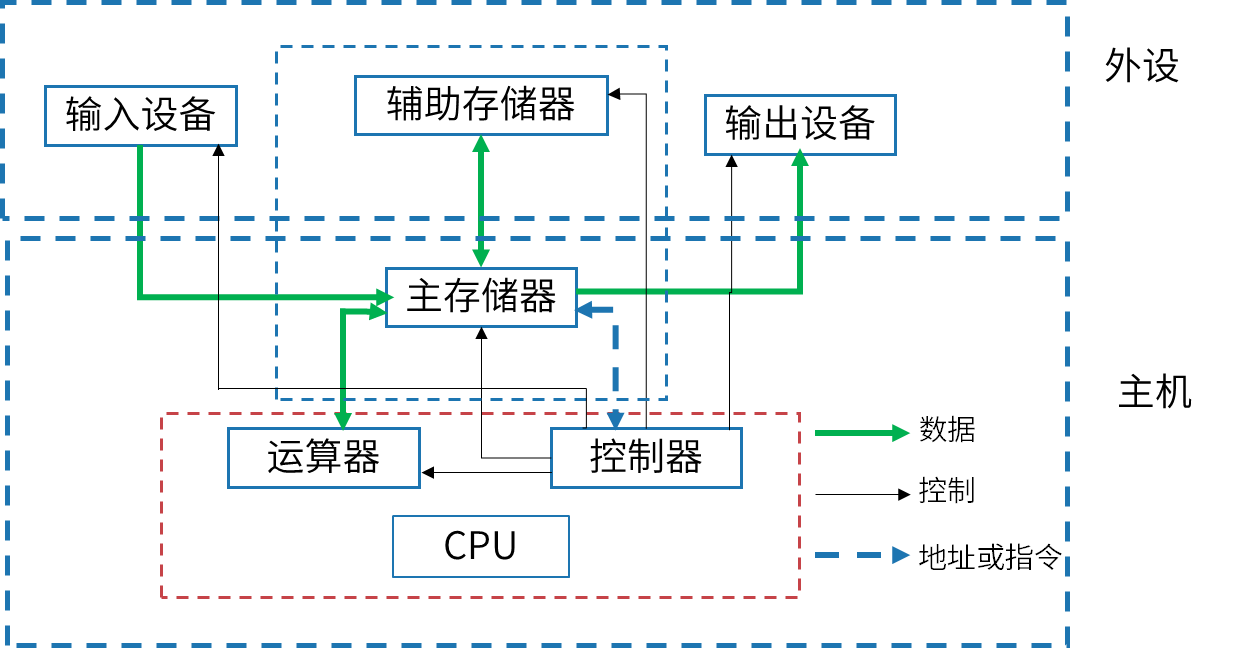
## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 计算机体系结构分类  （Flynn分类法） | Flynn分类法 |
| 1 | CISC与RISC（⭐） | CISC与RISC（CISC：复杂指令集；RISC：精简指令集。） |
| 1 | 存储系统（⭐⭐⭐⭐） | 分级存储体系 |
| 2 | Cache |
| 3 | 主存/内存 |
| 4 | 磁盘结构与参数 |
| 1 | 流水线（⭐⭐） | 流水线 |
| 1 | 校验码（⭐） | 校验码 |

## 2 考点精讲

### 2.1 计算机组成



CPU组成

（1）运算器

算术逻辑单元ALU：数据的算术运算和逻辑运算

累加寄存器AC：通用寄存器，为ALU提供一个工作区，用在暂存数据

数据缓冲寄存器DR：写内存时，暂存指令或数据

状态条件寄存器PSW：存状态标志与控制标志

（争议：也有将其归为控制器的）

（2）控制器

程序计数器PC：存储下一条要执行指令的地址

指令寄存器IR：存储即将执行的指令

指令译码器ID：对指令中的操作码字段进行分析解释

时序部件：提供时序控制信号

### 2.2 计算机体系结构

#### 2.2.1分类（Flynn分类法）

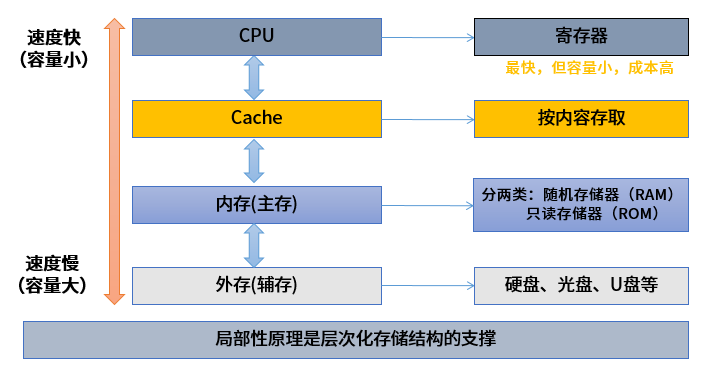
|  |  |  |  |
| --- | --- | --- | --- |
| 体系结构类型 | 结构 | 关键特性 | 代表 |
| 单指令流单数据流  SISD | 控制部分：一个  处 理 器：一个  主存模块：一个 |  | 单处理器系统 |
| 单指令流多数据流  SIMD | 控制部分：一个  处 理 器：多个  主存模块：多个 | 各处理器以异步的形式执行同一条指令 | 并行处理机  阵列处理机  超级向量处理机 |
| 多指令流单数据流  MISD | 控制部分：多个  处 理 器：一个  主存模块：多个 | 被证明不可能，至少是不实际 | 目前没有，有文献称流水线计算机为此类 |
| 多指令流多数据流  MIMD | 控制部分：多个  处 理 器：多个  主存模块：多个 | 能够实现作业、任务、指令等各级全面并行 | 多处理机系统  多计算机 |

#### 2.2.2 指令系统分类--CISC与RISC（⭐）（CISC：复杂指令集；RISC：精简指令集。）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令系统类型 | 指令 | 寻址方式 | 实现方式 | 其它 |
| CISC（复杂） | 数量多，使用频率差别大，可变长格式 | 支持多种 | 微程序控制技术（微码） | 研制周期长 |
| RISC（精简） | 数量少，使用频率接近，定长格式，大部分为单周期指令，操作寄存器，只有Load/Store操作内存 | 支持方式少 | 增加了通用寄存器；硬布线逻辑控制为主；适合采用流水线 | 优化编译，有效支持高级语言 |

### 2.3 存储系统（⭐⭐⭐⭐）

#### 2.3.1 分级存储体系概念



（1）关于多级存储体系的分布；

（2）关于多级存储体系不同层次的大小、速度、成本等对比；

（3）关于多级存储体系的理论支持体系（局部性原理）。

时间局部性：指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，典型原因是由于程序中存在着大量的循环操作。

空间局部性：指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将被访问，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，其典型情况是程序顺序执行。

工作集理论：工作集是进程运行时被频繁访问的页面集合。

#### 2.3.2 Cache

（1）Cache的相关概念（理论依据（局部性原理），大小、速度、成本等对比）

Cache的功能:提高CPU数据输入输出的速率，突破冯·诺依曼瓶颈，即CPU与存储系统间数据传送带宽限制。

在计算机的存储系统体系中，Cache是（除寄存器以外）访问速度最快的层次。

使用Cache改善系统性能的依据是程序的局部性原理。

如果以h代表对Cache的访问命中率，t1表示Cache的周期时间，t2表示主存储器周期时间，以读操作为例，使用“Cache+主存储器”的系统的平均周期为t3，则：t3 = h×t1+(1-h)×t2

其中，（1-h）又称为失效率（未命中率）。

（2）Cache映射方式

直接相联映像：硬件电路较简单，但冲突率很高。

全相联映像：电路难于设计和实现，只适用于小容量的cache，冲突率较低。

组相联映像：直接相联与全相联的折中。

（3）Cache页面淘汰算法

随机算法

先进先出算法（FIFO）

近期最少使用算法（LRU）-- 计数器统计使用次数

（4）Cache的读写过程

写直达：同时写Cache与内存

写回：只写Cache，淘汰页面时，写回内存

标记法：只写入内存，并将标志位清0，若用到此数据，需要再次调取

#### 2.3.3 主存/内存

（1）分类

相联存储器：按内容存取，如Cache。

随机存取存储器

DRAM（Dynamic RAM，动态RAM）-SDRAM

SRAM（Static RAM，静态）

只读存储器

MROM（Mask ROM，掩模式ROM）

PROM（Programmable ROM，一次可编程 ROM）

EPROM（Erasable PROM，可擦除的 PROM）

闪速存储器（flash memory，闪存）

（2）计算

内存单元数计算：最大地址+1-最小地址

内存总容量：按字节编址，内存单元数\*8bit；按字编址，内存单元数\*机器字长。

已知芯片单位容量，求所用芯片的片数，总容量/单位容量；

已知所用芯片的片数，求取芯片单位容量，总容量/芯片片数。

#### 2.3.4 磁盘结构与参数

（1）存取时间=寻道时间+等待时间，寻道时间是指磁头移动到磁道所需的时间；等待时间为等待读写的扇区转到磁头下方所用的时间。有时还需要加上数据的传输时间。

（2）在处理过程中，如果有关于缓冲区的使用，需要了解对于单缓冲区每次只能被一个进程使用，即向缓冲区传输数据的时候不能从缓冲区读取数据，反之亦然。

（3）对于磁盘存储的优化，是因为磁头保持转动的状态，当读取数据传输或处理时，磁头会移动到超前的位置，需要继续旋转才能回到逻辑下一磁盘块，优化存储就是调整磁盘块的位置，让逻辑下一磁盘块放到磁头将要开始读取该逻辑块的位置。

（4）磁盘移臂调度算法：

先来先服务FCFS（谁先申请先服务谁）；

最短寻道时间优先SSTF（申请时判断与磁头当前位置的距离，谁短先服务谁）；

扫描算法SCAN（电梯算法，双向扫描）；

循环扫描CSCAN（单向扫描）。

### 2.4 流水线（⭐⭐）

（1）流水线 – 概念

流水线是指在程序执行时多条指令重叠进行操作的一种准并行处理实现技术。各种部件同时处理是针对不同指令而言的，它们可同时为多条指令的不同部分进行工作，以提高各部件的利用率和指令的平均执行速度。

（2）相关参数计算

（流水线执行时间计算、流水线吞吐率、流水线加速比、流水线效率）

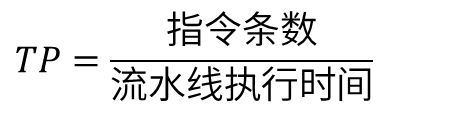
流水线建立时间：1条指令执行时间。

流水线周期：执行时间最长的一段

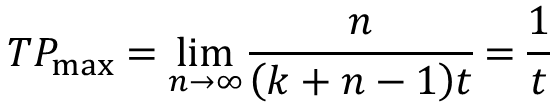
流水线执行时间（理论公式）：(t1+t2+..+tk)+(n-1)\*∆t

流水线执行时间（实践公式）：k\*∆t +(n-1)\*∆t

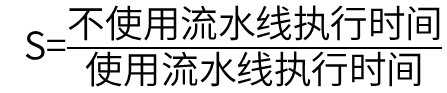
流水线吞吐率：



流水线最大吞吐率（即流水线周期的倒数）



流水线加速比：



### 2.5校验码（⭐）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 校验码位数 | 校验码位置 | 检错 | 纠错 | 校验方式 |
| 奇偶校验 | 1 | 一般拼接在头部 | 可检奇数位错 | 不可  纠错 | 奇校验：最终1的个数是奇数个；  偶校验：最终1的个数是偶数个； |
| CRC循环冗余校验 | 生成多项式最高次幂决定 | 拼接在信息位尾部 | 可检错 | 不可  纠错 | 模二除法求余数，拼接作为校验位 |
| 海明校验 | 2r≥m+r+1 | 插入在信息位中间（2k位置） | 可检错 | 可纠错 | 分组奇偶校验 |

## 3 章节问答

（1）移臂调度过程中，同一柱面有多个扇区怎么办？

答：

一般而言，涉及到移臂调度算法时，只需要考虑移臂过程，查看柱面顺序即可。旋转等待时间跟多磁头（盘面）是否并发读写有关系，从目前磁盘技术来看，是支持多个盘面同时读写的。 所以只要是同一柱面，扇区顺序没有要求。

（2）为什么说高速缓存对程序员来说是透明的？什么是透明？

答：

这里的透明指的是不需要了解其中的机制甚至是看不到这个层次，也就是不直接操作。高速缓存是由硬件自动完成的，所以对程序员来说是透明的。

（3）磁盘读取数据过程中单缓冲区和双缓冲区有什么区别？

答：

磁盘读取涉及到缓冲区一般形式如下：



处理数据可以理解为三个处理步骤：①从磁盘读入到缓冲区；②从缓冲区读入到（内存）用户区；③处理（内存）用户区数据。

如果是单缓冲区，则意味步骤①和②都需要访问临界资源——缓冲区，所以需要合并成一个操作阶段。（缓冲区同一时刻只允许一个进程访问）

如果是双缓冲区，可以实现读入到缓冲区2和从缓冲区1读入到用户区的并发。

可以构造流水线进行计算。

单缓冲区：使用缓冲区时不能并行，必须分开执行，时间为①和②的时间之和；然后处理数据。构造成流水线后，整个过程划分为2个阶段，分别是①和②的时间之和，③的时间，根据流水线执行公式进行计算。

双缓冲区：读入缓冲区，和从缓冲区读入用户区，可以对不同的缓冲区进行，也就是说，可以并行处理。对于这里构造成流水线后，整个过程划分为3个阶段，即①、②、③。根据流水线执行公式进行计算结课。

PS：注意理解这里流水线的构造过程。

（4）流水线执行时间计算有时间公式和理论公式，如何选择？

答：

一般而言，我们的计算都是以理论公式为准，如果在考试中理论公式找不到答案再带入实践公式。

# 第八章 嵌入式系统

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 嵌入式微处理器（⭐） | 嵌入式微处理器分类、嵌入式微处理器体系结构 |
| 2 | 总线（⭐） | 总线基本概念、总线分类 |
| 3 | 嵌入式系统软件（⭐） | 嵌入式系统软件特点、嵌入式系统软件分类、嵌入式系统体系结构 |
| 4 | 嵌入式操作系统（⭐⭐⭐⭐） | 嵌入式操作系统特点、嵌入式操作系统分类 |

**2 考点精讲**

### 2.1 嵌入式微处理器

#### 2.1.1嵌入式微处理器分类

（1）嵌入式微控制器（MCU：Micro Controller Unit）：又称为单片机，片上外设资源一般比较丰富，适合于控制。

（2）嵌入式微处理器（EMPU：Embedded Micro Processing Unit）： 又称为单板机，由通用计算机中的CPU发展而来，仅保留和嵌入式应用紧密相关的功能硬件。

（3）嵌入式DSP处理器（DSP：Digital Signal Processor）：专门用于信号处理方面的处理器。

（4）嵌入式片上系统（SOC）：追求产品系统最大包容的集成器件。

成功实现了软硬件的无缝结合，直接在微处理器片内嵌入操作系统的代码模块。

减小了系统的体积和功耗、提高了可靠性和设计生产效率。

#### 2.1.2嵌入式微处理器体系结构

（1）冯·诺依曼结构

冯·诺依曼结构也称普林斯顿结构，是一种将程序指令存储器和数据存储器合并在一起的存储器结构。

特点：

一般用于PC处理器，如I3，I5，I7处理器

指令与数据存储器合并在一起

指令与数据都通过相同的数据总线传输

（2）哈佛结构

哈佛结构是一种将程序指令存储和数据存储分开的存储器结构。哈佛结构是一种并行体系结构，它的主要特点是将程序和数据存储在不同的存储空间中，即程序存储器和数据存储器是两个独立的存储器，每个存储器独立编址、独立访问。

特点：

一般用于嵌入式系统处理器（DSP）

指令与数据分开存储，可以并行读取，有较高的数据吞吐率

有4条总线：指令和数据的数据总线与地址总线

### 2.2 总线

#### 2.2.1 总线的基本概念

基本概念

总线是一组能为多个部件分时共享的信息传送线，用来连接多个部件并为之提供信息交换通路。

特点：

挂接在总线上的多个部件只能分时向总线发送数据，但可同时从总线接收数据。

通过总线复用方式可以减少总线中信号线的数量，以较少的信号线传输更多的信息。

#### 2.2.2 总线分类

（1）从功能上来对总线进行划分：数据总线、地址总线和控制总线

数据总线（Data Bus，DB）：在CPU与RAM之间来回传送需要处理或是需要储存的数据。

地址总线（Address Bus，AB）：用来指定在RAM（Random Access Memory）之中储存的数据的地址。

控制总线（Control Bus，CB）：将微处理器控制单元（Control Unit）的信号传送到周边设备。

（2）从数据传输的方式划分为并行总线和串行总线

并行总线：将数据字节的各位用多条数据线同时进行传送。常见并行总线ISA、PCI、VME等。

串行总线：数据是一位一位地进行传输的，在传输中每一位数据都占据一个固定的时间长度。常见串行总线：RS232、SPI、I2C、USB、CAN、IEEE 1394等。

包括：半双工总线、全双工总线

### 2.3 嵌入式系统软件

#### 2.3.1 嵌入式系统软件基本概念

（1）基本概念

嵌入式系统是一种以应用为中心，以计算机技术为基础，可以适应不同应用对功能、可靠性、成本、体积和功耗等方面的要求，集可配置可裁剪的软、硬件于一体的专用计算机系统。

（2）嵌入式系统具有以下特点：

规模较小。

开发难度大。

硬件资源有限。

嵌入式软件一般涉及底层软件的开发，需要软、硬件基础。

开发环境和运行环境不同。

实时性和可靠性要求高。如火箭飞行控制、核电站。

要求固化存储。

#### 2.3.2 嵌入式系统软件分类

（1）根据系统对时间的敏感程度可将嵌入式系统划分为：

嵌入式系统、嵌入式实时系统（强实时系统、弱实时系统）

（2）从安全性要求看，嵌入式系统还可分为：

安全攸关系统、非安全攸关系统

#### 2.3.3 嵌入式系统软件的体系结构



设备驱动层（也叫板级支持包 BSP）

（1）概念

板级支持包（BSP）是介于主板硬件和操作系统中驱动层程序之间的一层，一般认为它属于操作系统的一部分，主要是实现对操作系统的支持，为上层的驱动程序提供访问硬件设备寄存器的函数包，使之能与硬件主板更好地运行。在嵌入式系统软件的组成中，就有BSP。BSP是相对于操作系统而言的，不同的操作系统对应不同定义形式的BSP,例如VxWorks的BSP和Linux的BSP相对于某一CPU来说尽管实现的功能一样，可是写法和接口定义是完全不同的，所以写BSP一定要按照该系统BSP的定义形式来写（BSP的编程过程大多数是在某一个成型的BSP模板上进行修改）。这样才能与上层OS保持正确的接口，良好地支持上层OS。

（2）功能

BSP主要功能为屏蔽硬件，提供操作系统及硬件驱动，具体功能包括：

单板硬件初始化，主要是CPU的初始化，为整个软件系统提供底层硬件支持；

为操作系统提供设备驱动程序和系统中断服务程序；

定制操作系统的功能，为软件系统提供一个实时多任务的运行环境；

初始化操作系统，为操作系统的正常运行做好准备。

（3）BSP特点

板级支持包一般包含相关底层硬件的初始化、数据的输入／输出操作和硬件设备的配置等功能，它主要具有以下两个特点。

硬件相关性，因为嵌入式实时系统的硬件环境具有应用相关性，而作为上层软件与硬件平台之间的接口，BSP需为操作系统提供操作和控制具体硬件的方法。

操作系统相关性，不同的操作系统具有各自的软件层次结构，因此不同操作系统具有特定的硬件接口形式。

### 2.4 嵌入式操作系统

#### 2.4.1 嵌入式操作系统特点

嵌入式操作系统具有一般操作系统的功能，同时具有嵌入式软件的特点，主要有：

非通用型操作系统；

在性能和实时性方面可能有严格的限制；

能源、成本和可靠性通常是影响设计的重要因素；

占用资源少；

可裁剪、可配置。

#### 2.4.2 嵌入式操作系统的分类

（1）按照系统对响应时间的敏感程度，可以分为：

硬实时系统：系统对响应时间有严格要求，若响应时间不能满足，是绝对不允许的，会引起系统的崩溃或致命的错误。

软实时系统：系统对响应时间有要求，若响应时间不能满足，会带来额外可接受的代价。

非实时系统：响应时间没有严格要求，如分时操作系统，基于公平性原则，各进程分享处理器，获得大致相同的运行时间。

（2）按软件体系结构，可以分为：

单体结构：是一种常见的组织结构。中间件和设备驱动程序通常集成在系统内核中。系统只有一个可执行文件，包含所有功能组件，功能模块可相互调用。如Linux。

分层结构：操作系统被划分为若干个层次（0～N），各个层次之间的调用关系是单向的，即某一层次上的代码只能调用比它低层的代码。如MS-DOS。

微内核结构：也称客户/服务（client/server）结构。在内核中，把操作系统的大部分功能都剥离出去，只保留最核心的功能单元（如进程管理、存储管理）。如VxWork。微内核体系结构如下图所示：



（3）微内核与单体内核对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | 实质 | 优点 | 缺点 |
| 单体内核 | 将图形、设备驱动及文件系统等功能全部在内核中实现，运行在内核状态和同一地址空间。 | 减少进程间通信和状态切换的系统开销，获得较高的运行效率。 | 内核庞大，占用资源较多且不易剪裁。  系统的稳定性和安全性不好。 |
| 微内核 | 只实现基本功能，将图形系统、文件系统、设备驱动及通信功能放在内核之外。 | 内核精练，便于剪裁和移植。  系统服务程序运行在用户地址空间，系统的可靠性、稳定性和安全性较高。  可用于分布式系统 | 用户状态和内核状态需要频繁切换，从而导致系统效率不如单体内核。 |

#### 2.4.3嵌入式实时操作系统

（1）嵌入式实时操作系统实时性的评价指标

中断响应和延迟时间

任务切换时间

信号量混洗时间

（2）嵌入式实时操作系统调度算法

优先级调度算法：系统为每个任务分配一个相对固定的优先顺序。

抢占式优先级调度算法：根据任务的紧急程度确定该任务的优先级。大多数RTOS调度算法都是抢占方式（可剥夺方式）。

最早截止期调度算法（EDF算法）：根据任务的截止时间来确定其优先级，对于时间期限最近的任务，分配最高的优先级。

最晚截止期调度算法

（3）常见的嵌入式RTOS（实时操作系统）：VxWorks、RT-Linux等。

|  |  |  |
| --- | --- | --- |
| 比较类型 | VxWorks | RT-Linux |
| 工作方式 | 操作系统与应用程序处于同一存储空间 | 操作系统与应用程序处于不同存储空间 |
| 多任务支持 | 支持多任务（线程）操作 | 支持多进程、多线程操作 |
| 实时性 | 实时系统 | 实时系统 |
| 安全性 | 任务间无隔离保护 | 支持进程间隔离保护 |
| 标准API | 支持 | 支持 |

## 3 章节问答

（1）微内核需要掌握到什么程度？

答：

微内核目前在架构考试中出现的主要是针对图示的考查。掌握微内核图示的各个部分，了解微内核与单体内核的区别。

（2）嵌入式操作系统需要掌握到什么程度？

答：

嵌入式是比较大的知识领域，从应试的角度来看，如果从事相关行业，以已有知识应试即可。如果对该领域并不熟悉，建议浅层次了解一些概念内容即可，不需要太过深入延伸。

# 第九章 操作系统

## 1 考情分析

### 1.1 本章概要

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 进程管理（⭐⭐⭐⭐） | 进程状态 |
| 2 | PV操作 |
| 3 | 前趋图 |
| 4 | 死锁 |
| 5 | 银行家算法 |
| 1 | 存储管理（⭐⭐⭐） | 页式存储/段式存储/段页式存储 |
| 2 | 页面置换算法 |
| 1 | 操作系统概念（⭐⭐） | 操作系统概念 |

## 2 考点精讲

### 2.1 操作系统概述

（1）软件分层



（2）操作系统作用

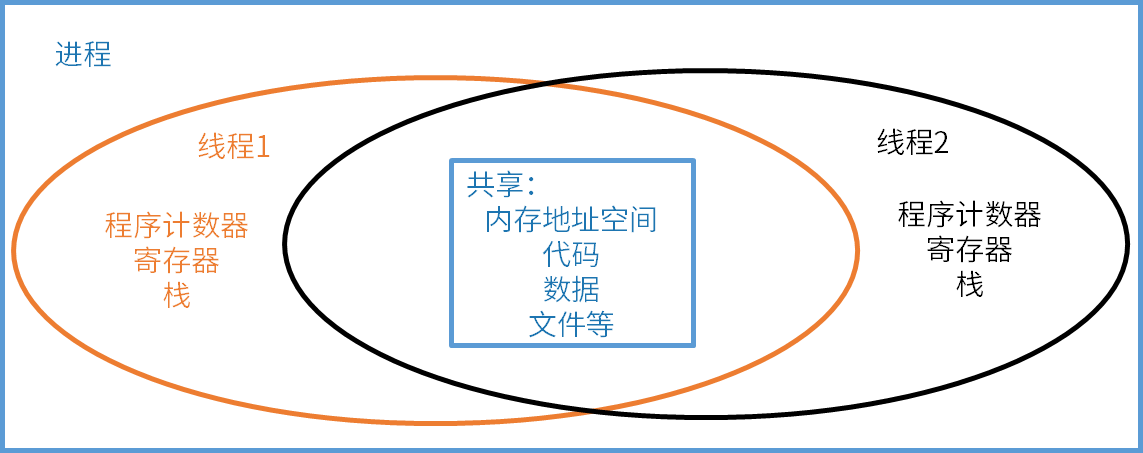
管理系统的硬件、软件、数据资源

控制程序运行

人机之间的接口

应用软件与硬件之间的接口

（3）线程



### 2.2 进程管理（⭐⭐⭐⭐）

#### 2.2.1 进程的状态（⭐）

（1）操作系统三态模型如下图所示：



运行：当一个进程在CPU上运行时。

（单处理机处于运行态的进程只有一个）

就绪：一个进程获得了除CPU外的一切所需资源，一旦得到处理机即可运行。

阻塞：阻塞也称等待或睡眠状态，一个进程正在等待某一事件发生（例如请求I/O、等待I/O完成等）而暂时停止运行，此时即使把CPU分配给进程也无法运行，故称进程处于阻塞状态。

（2）操作系统五态模型：



挂起原因：

进程过多，主存资源不足，此时必须将某些进程挂起，放到磁盘对换区，暂时不参与调度，以平衡系统负载；

系统出现故障，或者是用户调试程序，也可能需要将进程挂起检查问题。

#### 2.2.2 前趋图（⭐）

（1）前趋活动和后继活动：在前趋图中，前趋活动完成后通知所有后继活动；后继活动开始之前要检查是否前趋活动已经全部完成。

#### 2.2.3 信号量与PV操作（⭐⭐）

相关概念：互斥、同步、临界资源、临界区、信号量。

互斥：如千军万马过独木桥，同类资源的竞争关系。

同步：速度有差异，在一定情况停下等待，进程间的协作关系。

临界资源：诸进程间需要互斥方式对其进行共享的资源，如打印机、 磁带机等

临界区：每个进程中访问临界资源的那段代码称为临界区

信号量：是一种特殊的变量

PV操作对应的过程（如下图所示）：



前趋图与PV操作结合，根据前趋图箭线标注信号量，再根据进程图填空。

#### 2.2.4 死锁及银行家算法（⭐）

了解死锁的条件和预防概念：进程管理是操作系统的核心，但如果设计不当，就会出现死锁的问题。如果一个进程在等待一件不可能发生的事，则进程就死锁了。而如果一个或多个进程产生死锁，就会造成系统死锁。

根据题干给出的进程和资源分配，判断形成死锁的最小资源数或其他参数：对于这种情况，分配资源时每个进程得到可以完成进程的资源数减一，此时是形成死锁的最差情况，在此情况下多1个资源即可解决死锁问题，即不可能形成死锁。

银行家算法：当一个进程对资源的最大需求量不超过系统中的资源数时可以接纳该进程。

进程可以分期请求资源，但请求的总数不能超过最大需求量。

当系统现有的资源不能满足进程尚需资源数时，对进程的请求可以推迟分配，但总能使进程在有限的时间里得到资源。

根据银行家算法判断相关进程序列是否会形成死锁，是则为不安全序列。

### 2.3 存储管理（⭐⭐）

#### 2.3.1 段页式存储（⭐⭐）

（1）知道页面大小时，可以依此判断页内地址的长度，并据此知道该地址的页号；

（2）页号与页帧号的转换可以通过查表进行；

（3）段地址的格式，段号后跟段内地址不能超过段长；

（4）页式存储：将程序与内存均划分为同样大小的块，以页为单位将程序调入内存。

（5）段式存储：按用户作业中的自然段来划分逻辑空间，然后调入内存，段的长度可以不一样。

（6）段页式存储：段式与页式的综合体。先分段，再分页。1个程序有若干个段，每个段中可以有若干页，每个页的大小相同，但每个段的大小不同。

（7）快表是一块小容量的相联存储器（Associative Memory），由高速缓存器组成，速度快，并且可以从硬件上保证按内容并行查找，一般用来存放当前访问最频繁的少数活动页面的页号。

快表：将页表存于Cache上；慢表：将页表存于内存上。

#### 2.3.2 页面置换算法（⭐）

（1）页面淘汰时，主要依据原则：先淘汰最近未被访问的（访问位为0），其次淘汰但未被修改的（即修改位为0，因为修改后的页面）。

（2）页面淘汰算法有多种，常用的是LRU即最近最少使用原则，依据的是局部性原理。

（3）对于多种淘汰算法：最优算法OPT（理想型），随机算法RAND（随机性），先进先出FIFO（可能产生“抖动”），最近最少使用LRU（依据局部性原理）。

## 3 章节问答

（1）PV操作中信号量表示什么含义？

答：

PV操作中，信号量一般可以表示资源的数量。当信号量小于0，表示资源数不足，此时信号量可以表示等待资源的进程数量。

（2）PV操作中V操作能不能先于P操作出现？

答：

PV操作一定是成对出现，但是没有限定先后，也就是说，可以先进行V操作，此时只是释放资源，没有进程阻塞则不需要唤醒其他进程。在需要的地方再进行P操作。

（3）PV操作中如何区分互斥信号量和同步信号量？

答：

信号量一般与PV操作结合在一起，对于同步模型的描述中，涉及的就是同步信号量，一般同步是多个进程之间通过PV操作进行制约，此时P操作和成对的V操作一般是位于不同的进程中的，而互斥模型中PV操作控制的是对资源的访问限制，此时P操作和成对的V操作一般出现在资源使用前和适用后，一般位于同一个进程中。

（4）页式存储中页内地址长度如何判断？

答：

页式存储调用方式中，一般会给出页面的大小。依据计算机组成与体系结构章节中讲到的主存编址计算中，如果没有明确指出，则默认按字节编址。假设页面大小为512字节，则此时，默认按字节编址，存储单元个数为512个，地址编号为0-511，需要9位二进制对页内地址编号即为页内地址长度。如果给出页面大小为1K，此时指的就是存储单元个数，地址编号为0-1023，需要10位二进制作为页内地址长度对页内地址进行编号。

（5）位示图计算过程中，编号是从0开始还是从1开始？

答：

根据近几年的其他科目真题来看，目前考查时会给出编号形式是从0还是从1开始。如果没有给出，则默认位从0编号，字从1编号。

# 第十章 数据库系统

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 数据库设计阶段（⭐⭐） | 数据库设计阶段 |
| 1 | 关系代数（⭐⭐⭐⭐） | 关系代数运算 |
| 2 | 关系代数与SQL结合考查 |
| 3 | Amstrong公理 |
| 1 | 规范化理论（⭐⭐⭐⭐） | 候选关键字 |
| 2 | 范式 |
| 3 | 模式分解 |
| 4 | 其他 |
| 1 | 并发控制（⭐） | 并发控制 |
| 1 | 数据备份与恢复（⭐） | 数据备份与恢复 |
| 1 | 其他（⭐） | 其他 |

## 2 考点精讲

### 2.1 数据库模式

#### 2.1.1 体系结构

（1）三级模式：外模式对应视图，模式（也称为概念模式）对应数据库表，内模式对应物理文件。

（2）两层映像：外模式-模式映像，模式-内模式映像；两层映像可以保证数据库中的数据具有较高的逻辑独立性和物理独立性。

（3）物理独立性：即数据库的内模式发生改变时，应用程序不需要改变。

（4）逻辑独立性：即逻辑结构发生改变时，用户程序不需要改变。（逻辑独立性比物理独立性更难实现）



#### 2.1.2 视图

数据库视图

它一个虚拟表（逻辑上的表），其内容由查询定义（仅保存SQL查询语句）。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并没有真正存储这些数据，而是通过查询原始表动态生成所需要的数据。

视图的优点：

视图能简化用户操作

视图使用户能以多种角度看待同一数据

视图对重构数据库提供了一定程度的逻辑独立性

视图可以对机密数据提供安全保护

物化视图

它不是传统意义上虚拟视图，是实体化视图，其本身会存储数据。同时当原始表中的数据更新时，物化视图也会更新。

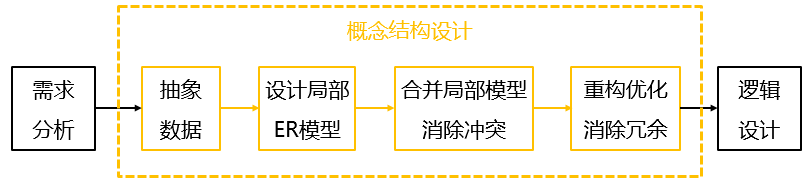
### 2.2 数据库设计过程



#### 2.2.1 需求分析过程

确认需求、确定设计目标 🡪 分析和收集数据 🡪 整理文档

#### 2.2.2 概念结构设计过程



E-R图集成的方法：

多个局部E-R图一次集成。

逐步集成，用累加的方式一次集成两个局部E-R。

集成产生的冲突及解决办法：

属性冲突：包括属性域冲突和属性取值冲突。

命名冲突：包括同名异义和异名同义。

结构冲突：包括同一对象在不同应用中具有不同的抽象，以及同一实体在不同局部E-R图中所包含的属性个数和属性排列次序不完全相同。

#### 2.2.3 逻辑结构设计

任务

1、确定数据模型；

2、将E-R图转换成指定的数据模型；

E-R模型转关系模式原则如下：

一个实体型转换为一个关系模式。

联系转关系模式：

1：1联系：可将联系合并至任意一端的实体关系模式中。

1：n联系：可将联系合并至n端实体关系模式中。

m：n联系：联系必须单独转成关系模式。

3、确定完整性约束；

4、确定用户视图。

设计过程

将概念结构向一般关系模型转换；

将第一步的得到的结构向特定的DBMS支持下的数据模型转换；

依据应用的需求和具体的DBMS的特征进行调整和完善。

#### 2.2.4 物理结构

任务

确定数据分布；

确定存储结构；

确定存取方式：存储记录结构设计；存储记录布局；存储方法的设计。

设计目标

数据库的物理设计是完全依赖于给定的硬件环境和数据库产品的，没有通用的物理设计方法可遵循，只能给出一般的设计内容和原则。

数据库的存储模式的设计可以不必考虑用户理解的方便，其设计目标是提高数据库的性能，其次是节省存储空间。

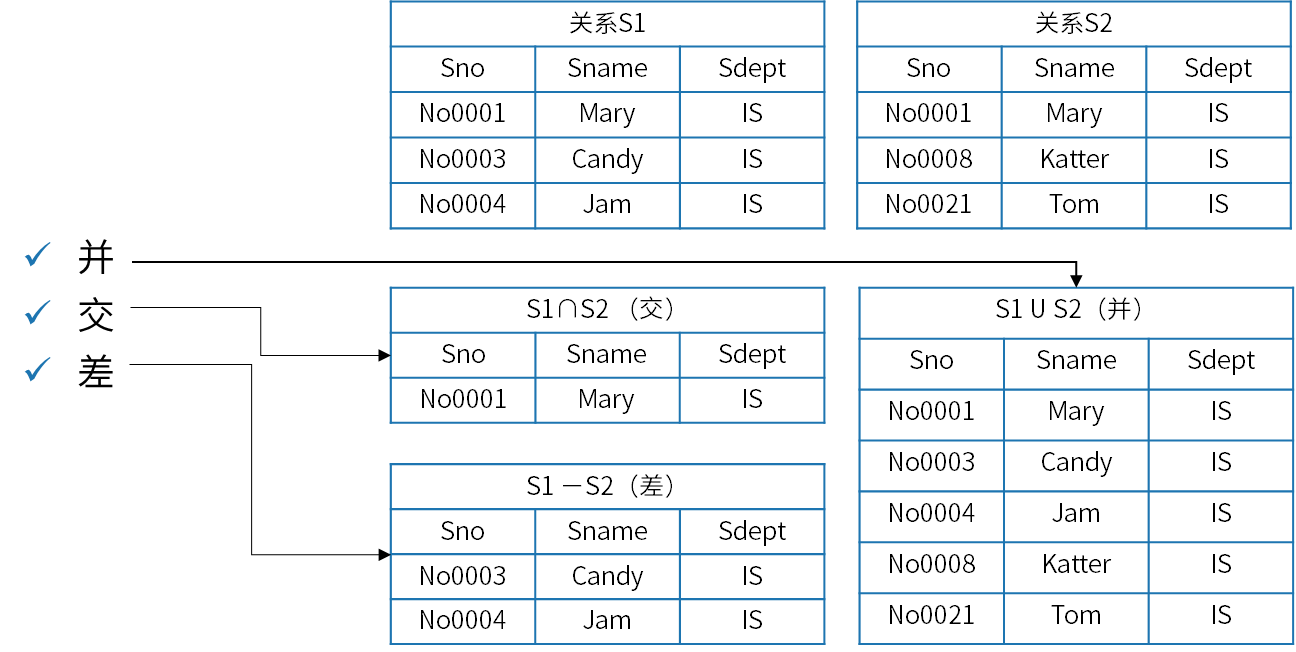
### 2.3 关系代数

#### 2.3.1 并（结果为二者元组之和去除重复行）

#### 2.3.2 交（结果为二者重复行）

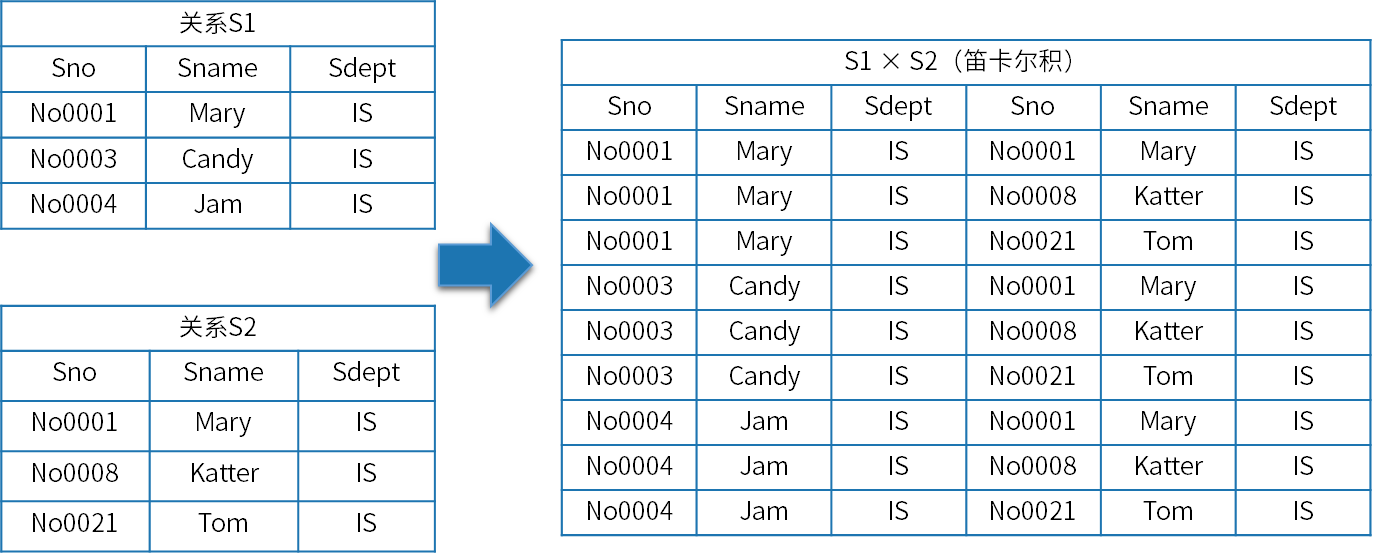
#### 2.3.3 差（前者去除二者重复行）

类似于集合运算，计算如下图所示：



#### 2.3.4 笛卡尔积

结果的属性列数为二者属性列数之和，结果的元组行数为二者元素数乘积。



#### 2.3.5 投影

对属性列的选择列出。

#### 2.3.6 选择

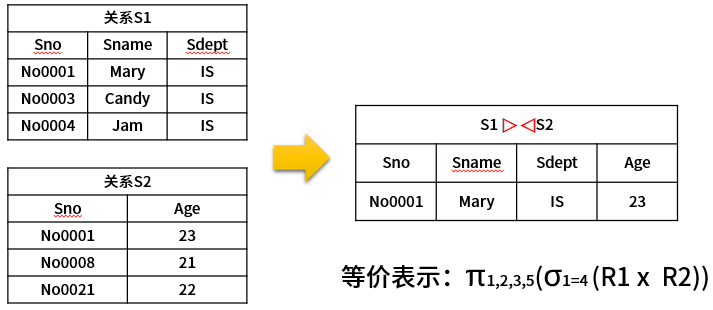
对元组行的选择列出。

属性名可以依次标序号，直接以数字形式出现在表达式中。计算如下图所示：

#### 2.3.7 自然连接

结果列数为二者属性列数之和减去重复列，行数为二者同名属性列其值相同的结果元组。笛卡尔积、选择、投影的组合表示可以与自然连接等价。

普通连接的条件会写出，没有写出则表示为自然连接。计算如图所示：



### 2.4 规范化理论

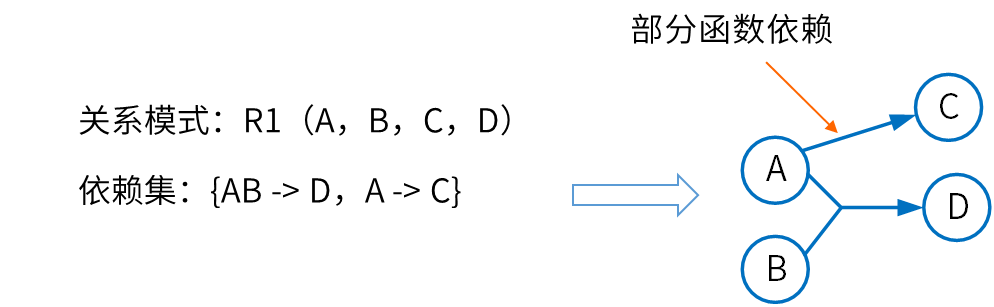
#### 2.4.1 非规范化存在的问题

规范化过程是为了解决数据冗余、删除异常、插入异常、更新异常等问题。

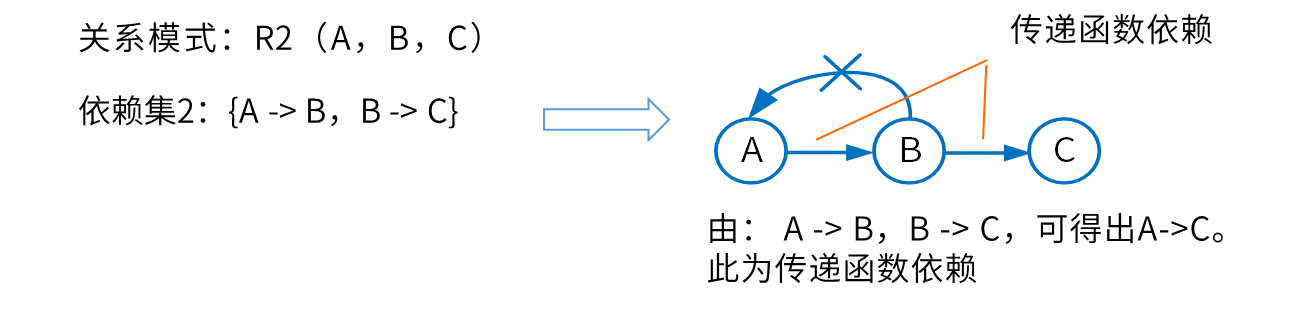
#### 2.4.2相关概念

函数依赖

部分函数依赖



传递函数依赖



键/码

候选键/候选码/码 ：如果在一个关系中，存在一个属性（或属性组）都能用来唯一标识该关系的元组，并不含有多余的属性，这个属性（或属性组）称为该关系的候选码或候选键。候选键可以是单属性也可以是多属性集合，候选键可以是一个也可以有多个。

候选键的求取可以利用图示法找入度为0的属性集合，并在此基础上进行扩展，最终找到能够遍历全图的最小属性组合作为候选键，对于入度为0在关系依赖集中可以理解为从未在箭线右侧出现。

全码：关系中的所有属性组成这个关系模式的候选码。

主键/主码：可以有多个不同的候选键，在其中任选一个作为主键。其值能惟一地标识元组的一个或多个属性，称为主码或关键字

组成候选码的属性就是主属性，其他为非主属性。

外键：关系中的某个属性（或属性组）不是该关系的主码或只是主码的一部分，但却是另一个关系的主码时，该属性（或属性组）称为这个关系的外码。

主属性/非主属性：关系中包含在一个候选码中的属性称为主属性或码属性，不包含在任何一个候选码中的属性称为非主属性或非码属性。

#### 2.4.3 Amstrong公理

关系模式R <U，F >来说有以下的推理规则：

自反律（Reflexivity）：若Y⊆X⊆U，则X →Y成立。

增广律（Augmentation）：若Z⊆U且X→Y，则XZ→YZ成立。

传递律（Transitivity）：若X→Y且Y→Z，则X→Z成立。

根据A1，A2，A3这三条推理规则可以得到下面三条推理规则：

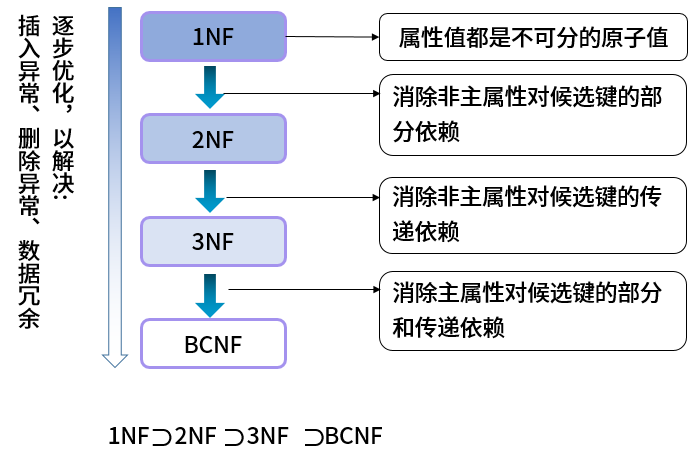
合并规则：由X→Y，X→Z，有X→YZ。 （A2， A3）

伪传递规则：由X→Y，WY→Z，有XW→Z。 （A2， A3）

分解规则：由X→Y及 Z ⊆ Y，有X→Z。 （A1， A3）

#### 2.4.4 范式

范式的关系



第一范式（1NF）：在关系模式R中，当且仅当所有域只包含原子值，即每个属性都是不可再分的数据项，则称关系模式R是第一范式。

第二范式（2NF）--消除非主属性对码的部分函数依赖：当且仅当关系模式R是第一范式（1NF），且每一个非主属性完全依赖候选键（没有不完全依赖）时，则称关系模式R是第二范式。

第三范式（3NF）--消除非主属性对码的传递函数依赖：当且仅当关系模式R是第二范式（2NF），且R中没有非主属性传递依赖于候选键时，则称关系模式R是第三范式。

BC范式（BCNF）--根据定义判断：设R是一个关系模式，F是它的依赖集，R属于BCNF当且仅当其F中每个依赖的决定因素必定包含R的某个候选码。

（可以理解为在3NF基础上，消除主属性之间的传递函数依赖和部分函数依赖）

#### 2.4.5 规范化过程-模式分解/拆表标准

无损分解：指将一个关系模式分解成若干个关系模式后，通过自然联接和投影等运算仍能还原到原来的关系模式

【公式法】

定理：如果R的分解为ρ＝{ R1，R2 }，F为R所满足的函数依赖集合，分解ρ具有无损联接性的充分必要条件是：

R1∩R2→（ R1－R2 ）

或 R1∩R2→（ R2－R1 ）

其中， R1∩R2表示模式的交，为R1与R2中公共属性组成， R1－R2或R2－R1表示模式的差集， R1－R2表示R1中去除R1和R2的公共属性所组成。当模式R分解成两个关系模式R1和R2时，如果R1与R2的公共属性能函数决定R1中或R2中的其它属性，这样的分解就具有无损联接性。

【表格法】

（1）构造初始表

（2）判定表构造

（3）观察判定表

#### 2.4.6 反规范化技术

概念

由于规范化会使表不断的拆分，从而导致数据表过多。这样虽然减少了 数据冗余，提高了增、删、改的速度 但会增加查询的工作量。系统需要进行多次连接，才能进行查询操作，使得系统的效率大大的下降。

技术手段

增加冗余列是指在多个表中具有相同的列，它常用来在查询时避免连接操作。

增加派生列指增加的列来自其它表中的数据，由它们计算生成。

重新组表指如果许多用户需要查看两个表连接出来的结果数据，则把这两个表重新组成一个表来减少连接而提高性能。例如，用户经常需要同时查看课程号，课程名称，任课教师号，任课教师姓名，则可把表class和表teacher合并成一个表，但需要更多的磁盘空间，同时也损失了数据在概念上的独立性。

分割表：水平分割与垂直分割。水平分割，有些记录常要查询，有些记录不常用，如历史记录。垂直分割，把主码与某些常用的字段组成一个表，把主码与另一些字段组成另一个表。

### 2.5 并发控制

#### 2.5.1 事务的特性（ACID）

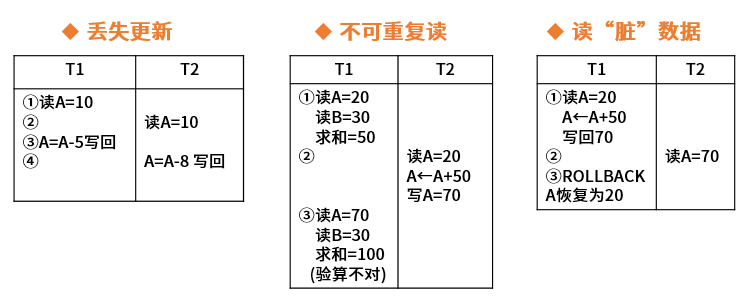
原子性（Atomicity）：整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性（Consistency）：一个事务可以封装状态改变（除非它是一个只读的）。事务必须始终保持系统处于一致的状态，不管在任何给定的时间并发事务有多少。

隔离性（Isolation）：隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请求，使得在同一时间仅有一个请求用于同一数据。

持久性（Durability）：在事务完成以后，该事务对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

#### 2.5.2 并发产生的问题



（1）丢失更新/丢失修改

（2）不可重复读

（3）读“脏”数据

#### 2.5.3 封锁技术

排他型封锁（简称X锁，也称为写锁）：如果事务T对数据A（可以是数据项、记录、数据集、乃至整个数据库）实现了X封锁，那么只允许事务T读取和修改数据A，其他事务不能对其加锁和操作。

共享型封锁（简称S锁，也称为读锁）：如果事务T对数据A实现了S封锁，那么只允许事务T读取数据A，不能修改数据A，其他事务可以读取数据A，并且可以对数据A加S锁，但不能加X锁。

#### 2.5.4 封锁协议

一级封锁协议。事务T在修改数据R之前必须先对其加X锁，直到事务结束才释放。可防止丢失修改。

二级封锁协议。一级封锁协议加上事务T在读取数据R之前先对其加S锁，读完后即可释放S锁。可防止丢失修改，还可防止读“脏”数据。

三级封锁协议。一级封锁协议加上事务T在读取数据R之前先对其加S锁，直到事务结束才释放。可防止丢失修改、防止读“脏”数据与防止数据重复读。

两段锁协议。可串行化的。可能发生死锁。

### 2.6 数据库完整性约束

防止的是对数据的意外破环。

实体完整性：规定基本关系R的主属性A不能取空。

参照完整性/引用完整性：规定，若F是基本关系R的外码，它与基本关系S的主码K，相对应（基本关系R和S不一定是不同的关系），则R中每个元组在F上的值必须为：或者取空值；或者等于S中某个元组的主码值。

用户自定义完整性：就是针对某一具体关系数据库的约束条件，反映某一具体应用所涉及的数据必须满足的语义要求，由应用的环境决定。如：年龄必须为大于0小于150的整数。

触发器：脚本编程

### 2.7 备份

|  |  |  |
| --- | --- | --- |
| 优缺点  备份方式 | 优点 | 缺点 |
| 冷备份 | 非常快速的备份方法(只需复制文件)；容易归档(简单复制即可)；容易恢复到某个时间点上(只需将文件再复制回去)；能与归档方法相结合，做数据库“最佳状态”的恢复 ；低度维护，高度安全 | 单独使用时，只能提供到某一时间点上的恢复；在实施备份的全过程中，数据库必须要作备份而不能做其他工作；若磁盘空间有限，只能复制到磁带等其他外部存储设备上，速度会很慢；不能按表或按用户恢复 |
| 热备份 | 可在表空间或数据库文件级备份，备份的时间短；备份时数据库仍可使用；可达到秒级恢复(恢复到某一时间点上)；可对几乎所有数据库实体做恢复；恢复是快速的 | 不能出错，否则后果严重；若热备份不成功，所得结果不可用于时间点的恢复；因难于维护，所以要特别小心，不允许“以失败告终” |

（1）冷备份也称为静态备份，是将数据库正常关闭，在停止状态下，将数据库的文件全部备份（复制）下来。

（2）热备份也称为动态备份，是利用备份软件，在数据库正常运行的状态下，将数据库中的数据文件备份出来。

（3）完全备份：备份所有数据

（4）差量备份：仅备份上一次完全备份之后变化的数据

（5）增量备份：备份上一次备份之后变化的数据

（6）日志文件 ：事务日志是针对数据库改变所做的记录，它可以记录针对数据库的任何操作，并将记录结果保存在独立的文件中。

## 3 章节问答

（1）什么是候选关键字？主键和候选键有什么关系？

答：

候选关键字是可以唯一标示元组且无冗余的属性集合。可以是单个属性，也可以是多个属性的集合，并且候选关键字可以有多个。主键是从候选键中选择一个。类似于从多个候选人中决定最终人选作为主键。

（2）如何判断候选关键字？

答：

候选关键字是可以唯一标示元组也就意味着通过候选键集合，可以访问到所有的属性。对于无法推导得出的属性集合（图示法中入度为0的属性集合），一定包含在候选关键字中。可以利用图示法求取候选关键字。首先是从入度为0的集合出发尝试遍历全图，如果不行，加入中间属性（既有入度也有出度的属性），从集合尝试遍历全图，直到能够遍历全图为止。

（3）规范化程度如何判断？

答：

规范化程度的判断其实就是范式的判断。范式从1NF->2NF->3NF->BCNF，这是一个层层递进的判断过程。首先根据属性是否能够再分判断是否满足1NF；然后找到关系模式的候选键、主属性、非主属性，判断是否有非主属性对候选键的部分函数依赖，如果有，该关系模式最高只能达到1NF，如果没有，该关系模式最高只能达到2NF；然后判断是否有非主属性对候选键的传递函数依赖，如果有，该关系模式最高只能达到2NF，如果没有，该关系模式最高只能达到3NF；最后判断函数依赖集中的所有函数依赖，是否满足左侧决定因素都包含候选键，如果没有满足，则该关系模式最高只能达到3NF，如果满足，该关系模式最高能够达到BCN。

（4）规范化程度是否越高越好？什么是反规范化？

答：

规范化程度一般达到3NF即可，并不是越高越好。由于规范化会使表不断的拆分，从而导致数据表过多。这样虽然减少了 数据冗余，提高了增、删、改的速度 但会增加查询的工作量。系统需要进行多次连接，才能进行查询操作，使得系统的效率大大的下降。规范化设计后，数据库设计者希望牺牲部分规范化来提高性能，这种从规范化设计的回退方法称为反规范化技术。

# 第十一章 计算机网络

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | TCP/IP协议簇（⭐⭐⭐） | TCP与UDP协议 |
| DNS |
| DHCP |
| 其他 |
| 1 | 综合布线（⭐） | 综合布线 |
| 1 | IPv6（⭐） | IPv6地址分类 |

## 2 考点精讲

### 2.1 TCP/IP协议族（⭐⭐⭐）

#### 2.1.1 常见协议及功能



POP3：110端口，邮件收取

SMTP：25端口，邮件发送

FTP：20数据端口/21控制端口，文件传输协议

HTTP：80端口，超文本传输协议，网页传输

DHCP：67端口，IP地址自动分配

SNMP：161端口，简单网络管理协议

DNS：53端口，域名解析协议，记录域名与IP的映射关系

TCP：可靠的传输层协议

UDP：不可靠的传输层协议

ICMP：因特网控制协议，PING命令来自该协议

IGMP：组播协议

ARP：地址解析协议，IP地址转换为MAC地址

RARP：反向地址解析协议，MAC地址转IP地址

#### 2.1.2 TCP与UDP

|  |  |  |
| --- | --- | --- |
|  | TCP | UDP |
| 共同点 | 基于IP协议的传输层协议，可以端口寻址 | |
| 不同点 | 面向连接（连接管理）、三次握手、流量控制、差错校验和重传、IP数据报按序接收不丢失不重复、可靠性强、牺牲通信量、效率低 | 不可靠、无连接、错误检测功能弱，无拥塞控制、无流量控制，有助于提高传输的高速率性。  不对无序IP数据报重新排序、不负责重传、不消除重复IP数据报、不对已收到的数据报进行确认、不负责建立或终止连接，这些由UDP进行通信的应用程序进行处理。 |
| 相关协议 | HTTP、FTP、Telnet、POP3、SMTP | DNS、DHCP、TFTP、SNMP |

#### 2.1.3 DHCP用法



（1）客户机/服务器模型

（2）租约默认为8天

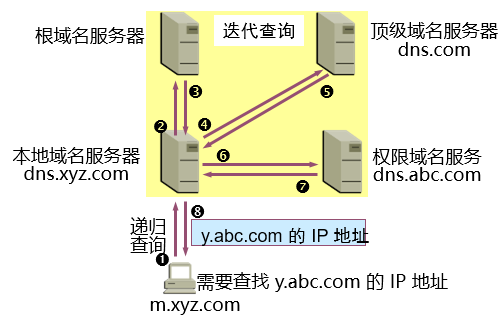
（3）当租约过半时，客户机需要向DHCP服务器申请续租；

（4）当租约超过87.5%时，如果仍然没有和当初提供IP的DHCP服务器联系上，则开始联系其他的DHCP服务器。

（5）固定分配、动态分配和自动分配。

（6）分配失败则IP显示为169.254.X.X（Windows） 和 0.0.0.0（Linux）

#### 2.2.4 DNS用法



（1）查询方式

递归查询：服务器必需回答目标IP与域名的映射关系。

迭代查询：服务器收到一次迭代查询回复一次结果，这个结果不一定是目标IP与域名的映射关系，也可以是其它DNS服务器的地址。

（2）查询过程

本机查询一般先查找本机HOST文件，没有相关映射时，查询域名服务器；

主机向本地域名服务器的查询一般采用的都是递归查询；

如果主机所询问的本地域名服务器不知道被查询域名的 IP 地址，那么本地域名服务器就以 DNS 客户的身份，向其他根域名服务器继续发出查询请求报文；

本地域名服务器向根域名服务器的查询通常采用迭代查询；

本地域名服务器向根域名服务器的查询通常是采用迭代查询。当根域名服务器收到本地域名服务器的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地域名服务器：“你下一步应当向哪一个域名服务器进行查询”。然后让本地域名服务器进行后续的查询。

根服务器或者流量较大的域名服务器都不使用递归查询，其原因也很简单，大量的递归查询会导致服务器过载；

### 2.2 IPv6（⭐⭐）

#### 2.2.1 概念

IPv6是设计用于替代现行版本IP协议（IPv4）的下一代IP协议。

（1）IPv6地址长度为128位，地址空间增大了296倍；

（2）灵活的IP报文头部格式。使用一系列固定格式的扩展头部取代了IPv4中可变长度的选项字段。IPv6中选项部分的出现方式也有所变化，使路由器可以简单路过选项而不做任何处理，加快了报文处理速度；

（3）IPv6简化了报文头部格式，字段只有8个，加快报文转发，提高了吞吐量；

（4）提高安全性。身份认证和隐私权是IPv6的关键特性；

（5）支持更多的服务类型；

（6）允许协议继续演变，增加新的功能，使之适应未来技术的发展；

#### 2.2.2 地址分类



IPv6规定每个网卡最少有3个IPv6地址，分别是链路本地地址、全球单播地址和回送地址，这些地址都可以是自动分配的。链路本地地址用于在链路两端传输数据，类似于（但不完全等同于）IPv4的私用IP地址。全球单播地址用于在Internet上传输数据，类似于IPv4中的合法的公网IP地址。回送地址用于网络测试，类似于IPv4的127.0.0.1。

单播地址(Unicast)：用于单个接口的标识符，传统的点对点通信。

组播地址(Multicast)：多播地址，一点对多点的通信，数据包交付到一组计算机中的每一个。IPv6没有广播的术语，而是将广播看做多播的一个特例。

任播地址(Anycast)：泛播地址，这是IPv6增加的一种类型。任播的目的站是一组计算机，但数据包在交付时只交付给其中一个，通常是举例最近的一个

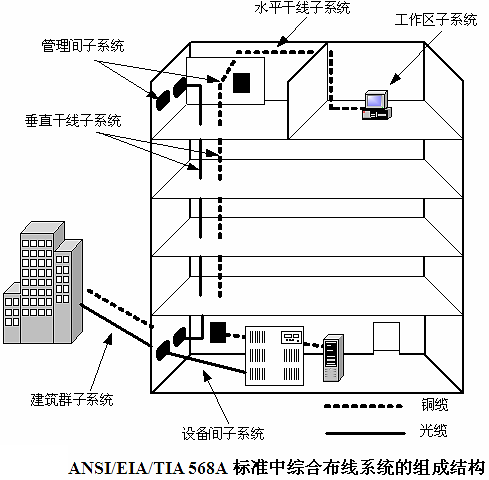
#### 2.2.3 IPv4/IPv6过渡技术

（1）双协议栈技术：双栈技术通过节点对IPv4和IPv6双协议栈的支持，从而支持两种业务的共存。

（2）隧道技术：隧道技术通过在IPv4网络中部署隧道，实现在IPv4网络上对IPv6业务的承载，保证业务的共存和过渡。隧道技术包括：6to4隧道；6over4隧道；ISATAP隧道。

（3）NAT-PT技术：NAT－PT使用网关设备连接IPv6和IPv4网络。当IPv4和IPv6节点互相访问时，NAT－PT网关实现两种协议的转换翻译和地址的映射。

### 2.3 综合布线系统（⭐）



（1）工作区子系统由信息插座、插座盒、连接跳线和适配器组成。

（2）水平子系统由一个工作区的信息插座开始，经水平布置到管理区的内侧配线架的线缆所组成。

（3）管理子系统由交连、互连配线架组成。管理子系统为连接其它子系统提供连接手段。

（4）垂直干线子系统由建筑物内所有的垂直干线多对数电缆及相关支撑硬件组成，以提供设备间总配线架与干线接线间楼层配线架之间的干线路由。

（5）设备间子系统是由设备间中的电缆、连接器和有关的支撑硬件组成，作用是将计算机、PBX、摄像头、监视器等弱电设备互连起来并连接到主配线架上。

（6）建筑群子系统将一个建筑物的电缆延伸到建筑群的另外一些建筑物中的通信设备和装置上，是结构化布线系统的一部分，支持提供楼群之间通信所需的硬件。它由电缆、光缆和入楼处的过流过压电气保护设备等相关硬件组成，常用介质是光缆。

## 3 章节问答

（1）TCP与UDP有什么区别？

答：

TCP采用可变大小的滑动窗口协议进行流量控制。在前向纠错系统中，当接收端检测到错误后就根据纠错编码的规律自行纠错；在后向纠错系统中，接收方会请求发送方重发出错分组。IP协议不预先建立虚电路，而是对每个数据报独立地选择路由并一站一站地进行转发，直到送达目标地。

（2）DNS查询的过程为什么有的是先查区域记录hosts，有的是先查本地缓存？

答：

本机（浏览器客户端）查询一般先查找本机HOST文件，没有相关映射时，查询域名服务器。域名服务器接收到请求时先查询本地缓存。

（3）计算机网络的考查为什么有些概念比较陌生？对于这些陌生的概念如何备考？

答：

计算机网络的考查覆盖范围比较零散，这一部分建议熟悉高频考点即可。

# 第十二章 数学与经济管理

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 图论（⭐） | 网络与最大流量 |
| 1 | 运筹方法（⭐⭐⭐） | 线性规划 |
| 2 | 动态规划 |
| 1 | 预测与决策（⭐） | 预测 |
| 2 | 决策 |
| 1 | 数学建模（⭐） | 数学建模概念 |

## 2 考点精讲

### 2.1 数学应用（⭐⭐⭐）

#### 【常考题型（文字无法详细介绍，可以查看相关视频和例题）】

#### 2.1.1 图论应用

最小生成树

最短路径

网络与最大流量

#### 2.1.2 运筹方法

线性规划（线性规划问题的数学模型通常由线性目标函数、线性约束条件、变量非负条件组成。）

动态规划

#### 2.1.3 预测

博弈论

状态转移矩阵

排队论

#### 2.1.4 决策

不确定型决策

决策树

### 2.2 数学建模（⭐）

#### 2.2.1 概念

数学建模是一种数学的思考方法，是运用数学的语言和方法，通过抽象和简化，建立能近似刻画并解决实际问题的模型的一种强有力的数学手段。

#### 2.2.2 建模过程

（1）模型准备：了解问题的实际背景，明确其实际意义，掌握对象的各种信息。用数学语言来描述问题。

（2）模型假设：根据实际对象的特征和建模的目的，对问题进行必要的简化，并用精确的语言提出一些恰当的假设。

（3）模型建立：在假设的基础上，利用适当的数学工具来刻划各变量之间的数学关系，建立相应的数学结构。只要能够把问题描述清楚，尽量使用简单的数学工具。

（4）模型求解：利用获取的数据资料，对模型的所有参数做出计算（估计）。

（5）模型分析：对所得的结果进行数学上的分析。

（6）模型检验：将模型分析结果与实际情形进行比较，以此来验证模型的准确性、合理性和适用性。如果模型与实际较吻合，则要对计算结果给出其实际含义，并进行解释。如果模型与实际吻合较差，则应该修改假设，再次重复建模过程。

（7）模型应用：应用方式因问题的性质和建模的目的而异。

#### 2.2.3 数学建模方法和思路

（1）直接分析法：认识原理，直接构造出模型。

（2）类比法：根据类似问题模型构造新模型。

（3）数据分析法：大量数据统计分析之后建模。

（4）构想法：对将来可能发生的情况给出设想从而建模。

## 3 章节问答

（1）数学与经济管理在架构考试中一般考的多吗？

答：

一般而言，架构设计师考试中会在上午的选择题中出现2分左右的数学与经济管理相关考查试题。下午题不会考查。占比并不多。

（2）数学与经济管理如何备考？

答：

这一部分内容，在考试中的规律性并不强，要求大家掌握给出的高频考点即可，其他的零散知识内容，不建议深入探索，可以在做题过程中，当做概念了解并记忆。

# 第十三章 系统配置与性能评价

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 性能评价方法（⭐⭐⭐） | 性能评价方法 |

## 2 考点精讲

### 2.1 性能评价方法（⭐⭐⭐）

（1）时钟频率法：以时钟频率高低衡量速度。

（2）指令执行速度法：表示机器运算速度的单位是MIPS。

（3）等效指令速度法 （ Gibson mix，吉普森混合法）：通过各类指令在程序中所占的比例（Wi）进行计算得到的。特点：考虑指令比例不同的问题。

（4）数据处理速率法（PDR）：PDR值的方法来衡量机器性能，PDR值越大，机器性能越好。PDR = L/R 特点：考虑CPU+存储

（5）综合理论性能法（CTP）：CTP用MTOPS（Million Theoretical Operations Per Second，每秒百万次理论运算）表示。CTP的估算方法是，首先算出处理部件每个计算单元的有效计算率，再按不同字长加以调整，得出该计算单元的理论性能，所有组成该处理部件的计算单元的理论性能之和即为CTP。

（6）基准程序法：把应用程序中用得最多、最频繁的那部分核心程序作为评估计算机系统性能的标准程序，称为基准测试程序（benchmark）。基准程序法是目前一致承认的测试系统性能的较好方法。

真实的程序->核心程序->小型基准程序->合成基准程序

## 3 章节问答

（1）阿姆达尔解决方案公式记不住怎么办？

答：

建议以例题的形式掌握解题过程即可。

（2）性能指标需要记住哪些？

答：

了解常见的计算机性能指标、系统性能指标、网络性能指标即可。

（3）性能评价方法如何掌握？

答：

基准程序测试的参照顺序要求熟悉。其他概念性内容注意区分记忆。

# 第十四章 知识产权与标准化

## 1 考情分析

### 1.1 本章重点

|  |  |  |
| --- | --- | --- |
| 序号 | 知识领域 | 知识点详情 |
| 1 | 保护范围与对象（⭐⭐⭐） | 保护范围与对象 |
| 1 | 保护期限（⭐） | 保护期限 |
| 1 | 知识产权人确定（⭐⭐⭐） | 知识产权人确定 |
| 1 | 侵权判断（⭐⭐⭐） | 侵权判断 |

## 2 考点精讲

### 2.1 知识产权

#### 2.1.1 保护对象和范围（⭐⭐⭐）

|  |  |  |
| --- | --- | --- |
| 法律法规名称 | 保护对象及范围 | 注意事项 |
| 著作权法 | 著作权  文学、绘画、摄影等作品 | 1、不需要申请，作品完成即开始保护  2、绘画或摄影作品原件出售（赠予）著作权还归原作者，原件拥有者有：所有权、展览权。 |
| 软件著作权法  计算机软件保护条例 | 软件著作权  软件作品 | 1、不需要申请，作品完成即开始保护  2、登记制度便于举证 |
| 专利法 | 专利权 | 需要申请，专利权有效期是从申请日开始计算 |
| 商标法 | 商标权 | 需要申请，核准之日起商标受保护 |
| 反不正当竞争法 | 商业秘密权 | 1、商业秘密包括技术与经营两个方面  2、必须有保密措施才能认定商业秘密 |

#### 2.1.2 保护期限（⭐⭐⭐）

|  |  |  |
| --- | --- | --- |
| 客体类型 | 权力类型 | 保护期限 |
| 公民作品 | 署名权、修改权、保护作品完整权 | 没有限制 |
| 发表权、使用权和获得报酬权 | 作者终生及其死亡后的50年（第50年的12月31日） |
| 单位作品 | 发表权、使用权和获得报酬权 | 50年（首次发表后的第50年的12月31日），若其间未发表，不保护。 |
| 公民软件产品 | 署名权、修改权 | 没有限制 |
| 发表权、复制权、发行权、出租权、信息网络传播权、翻译权、使用许可权、获得报酬权、转让权 | 作者终生及死后50年（第50年12月31日）。合作开发，以最后死亡作者为准。 |
| 单位软件产品 | 发表权、复制权、发行权、出租权、信息网络传播权、翻译权、使用许可权、获得报酬权、转让权 | 50年（首次发表后的第50年的12月31日），若其间未发表，不保护 |
| 注册商标 | | 有效期10年（若注册人死亡或倒闭1年后，未转移则可注销，期满后6个月内必须续注） |
| 发明专利权 | | 保护期为20年（从申请日开始） |
| 实用新型和外观设计专利权 | | 保护期为10年（从申请日开始） |
| 商业秘密 | | 不确定，公开后公众可用 |

#### 2.1.3 知识产权人确定（⭐⭐⭐⭐）

|  |  |  |  |
| --- | --- | --- | --- |
| 情况说明 | | 判断说明 | 归属 |
| 作品 | 职务作品 | 利用单位的物质技术条件进行创作，并由单位承担责任的 | 除署名权外其他著作权归单位 |
| 有合同约定，其著作权属于单位 | 除署名权外其他著作权归单位 |
| 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 | 单位享有著作权 |
| 属于从事本职工作活动的结果 | 单位享有著作权 |
| 使用了单位资金、专用设备、未公开的信息等物质、技术条件，并由单位或组织承担责任的软件 | 单位享有著作权 |
| 专利权 | 职务  作品 | 本职工作中作出的发明创造 | 单位享有专利 |
| 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 | 单位享有专利 |
| 离职、退休或调动工作后1年内，与原单位工作相关 | 单位享有专利 |

|  |  |  |  |
| --- | --- | --- | --- |
| 情况说明 | | 判断说明 | 归属 |
| 作品  软件 | 委托  创作 | 有合同约定，著作权归委托方 | 委托方 |
| 合同中未约定著作权归属 | 创作方 |
| 合作  开发 | 只进行组织、提供咨询意见、物质条件或者进行其他辅助工作 | 不享有著作权 |
| 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（除知名商标的非法抢注）  同时申请，则根据谁先使用（需提供证据）  无法提供证据，协商归属，无效时使用抽签（但不可不确定） | |
| 专利 | | 谁先申请谁拥有  同时申请则协商归属，但不能够同时驳回双方的专利申请 | |

#### 2.1.4 侵权判断（⭐⭐⭐⭐）

（1）适用情景

中国公民、法人或者其他组织的作品，不论是否发表，都享有著作权。

开发软件所用的思想、处理过程、操作方法或者数学概念不受保护

著作权法不适用于下列情形：

法律、法规，国家机关的决议、决定、命令和其他具有立法、行政、司法性质的文件，及其官方正式译文；

时事新闻；

历法、通用数表、通用表格和公式。

（2）常见侵权与合理引用区分

|  |  |
| --- | --- |
| 不侵权 | 侵权 |
| * 个人学习、研究或者欣赏； * 适当引用； * 公开演讲内容 * 用于教学或科学研究 * 复制馆藏作品； * 免费表演他人作品； * 室外公共场所艺术品临摹、绘画、摄影、录像； * 将汉语作品译成少数民族语言作品或盲文出版。 | * 未经许可，发表他人作品； * 未经合作作者许可，将与他人合作创作的作品当作自己单独创作的作品发表的； * 未参加创作，在他人作品署名； * 歪曲、篡改他人作品的； * 剽窃他人作品的； * 使用他人作品，未付报酬； * 未经出版者许可，使用其出版的图书、期刊的版式设计的。 |

## 3 章节问答

（1）在做真题时发现情景描述差不多，但侵权判断不一样，是答案不正确吗？

答：

对于知识产权的问题，经常会出现细节上的差异，注意仔细审题。比如对于盗版担责的问题，“某人持有盗版软件，但本人确实不知道是盗版的”和“某人持有盗版软件，但不知道该软件是盗版的，该软件的提供者不能证明其提供的复制品有合法来源”，前者只需要提供者担责，后者提供者和持有者都需要担责。后者的描述，有合理理由推论或者认定持有者应当知道其所使用运行的软件为侵权复制品。