1、分布式的定义与挑战、分布式系统的互联、网络与中间件

(1)分布式计算的定义与挑战

定义：分布式计算系统是由多个相互连接的计算机组成的一个整体，这些计算机在一组系统软件（分布式操作系统或中间件）环境下，合作执行一个共同的或不同的任务，最少依赖于集中的控制过程、数据和硬件。

分布式系统的要求：开放性、可扩展性、异构性、透明性

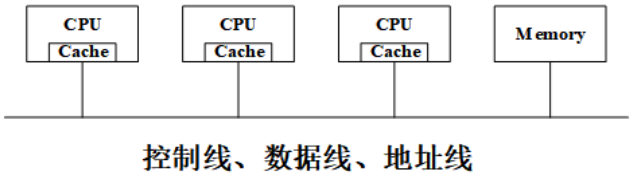
> 开放性：新的共享资源或服务能被加入并为各客户程序所利用。技术：统一的通讯协议、遵循公开的访问共享资源的标准化接口
>
> 可扩展性：规模、地域、管理可扩展。分布式系统设计以应对不断增长的规模；分布式系统可以在不同的地理位置部署其组件；分布式系统通过将不同的功能模块拆分为独立的服务或微服务，以及采用自动化工具和云服务，提高了管理的可扩展性
>
> 异构性：用户可以访问运行在异构计算机和网络上的服务，用户程序可以运行在这类异构的结构上。表现在：异构网络/计算机硬件/操作系统/程序设计语言、不同开发商的实现。技术：中间件技术/虚拟机技术。
>
> 透明性：向用户和应用程序隐藏了分布式计算系统部件的差异。规定了：访问/位置/并发/失效/复制/迁移/性能/规模透明性。
>
> 安全性：基于开发环境下的分布式计算系统应有安全措施，它们的安全至关重要。技术：身份认证技术/消息加密技术/访问控制技术。

(2)分布式计算系统的互联

同构多计算机系统

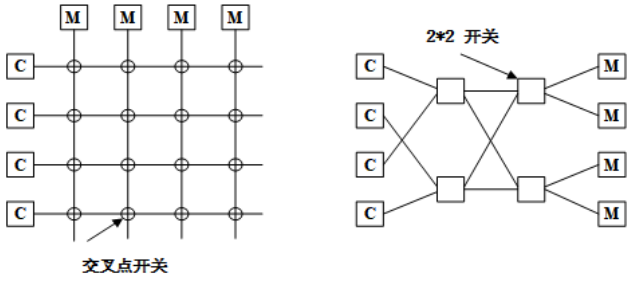
同构：计算机节点是相同的，它们使用相同的硬件架构、操作系统和软件环境

- 共享存储器：总线型多处理机



  存在问题：总线过载（解决：cache）、命中率、缓存一致性

- 共享存储器：交换型多处理机



交叉开关线：n^2个交叉开关点

Omega开关网：2 • 2开关点，共n ($\log_2 n$) /2个

异构多计算机系统

异构：组成异构多计算机的节点可能有着很大的差异，表现在处理机类型、存储器容量大小、I/O带宽以及操作系统等方面。有的节点本身就可能是多处理机系统、集群或并行高性能计算机。

通过互联网络（Internet）连接

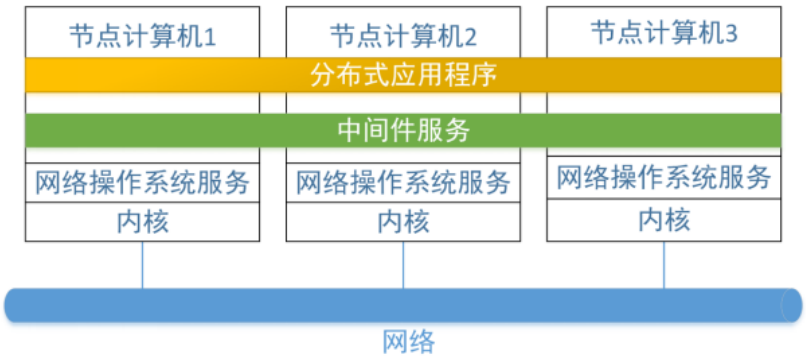
(3)分布式操作系统

- 微内模式：每个站点一个微内核，仅提供存储、进程IPC等核心管理功能和原语
- 集成式：每个站点上运行一个比较完整的OS
- 客户机/服务器：站点分为客户机和服务器，将计算和服务分布到不同的节点
- 中央式：有一个中央结点和若干卫星节点，每个节点上进程通过中央结点进行通信
- 分散式：DOS（磁盘操作系统）功能分散到一些节点，每个节点仅负责部分管理功能，需要各节点协商、合作方式进行管理

(4)网络与中间件

中间件（LSF，负载均衡系统）的概念：中间件运行在各节点的操作系统之上，节点计算机硬件和操作系统可以是异构的。中间件屏蔽了节点计算机的差异，为应用程序提供了统一的运行环境。

中间件提供的服务：命名服务、作业调度、高级通信服务、资源管理、数据的持久性、分布式事务、分布式文档系统、安全服务。



2、名字服务的概念，递归解析和迭代解析的工作流程及其优缺点

(1)名字服务的形式

名字服务（白页服务）

名字数据库是命名实体与其属性（地址）绑定的集合。名字服务器**根据名字实体的名字查找它的属性（地址）**。

目录服务（黄页服务）

基于属性描述查找实体。目录服务既可以根据实体的名字查找实体的属性，也可以根据实体的一个或多个属性及其值查找并得到一个匹配这些属性的实体列表。

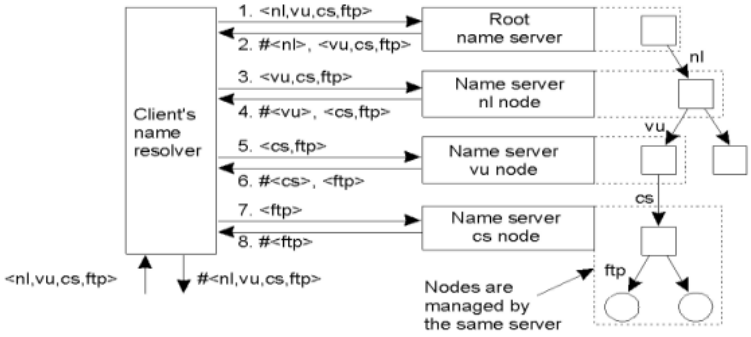
合约服务（绿页服务）

增强的目录服务，通过技术规范来定位一个命名实体。

(2)名字解析的实现

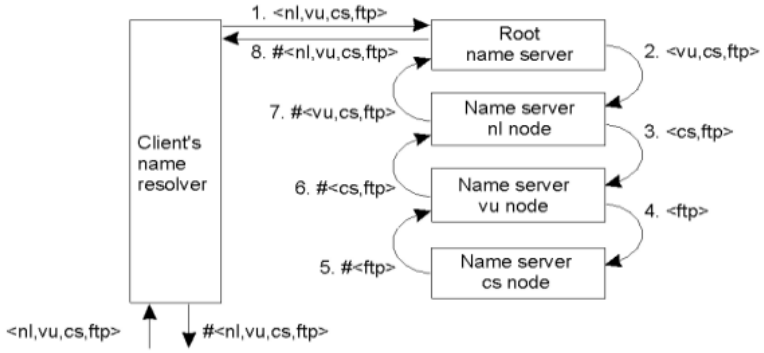
迭代解析

我不知道，问根，根告诉我A知道，我再问A，A告诉我B知道，我再问B.....



递归解析

我不知道，问根，根问知道的服务器A，A问知道的服务器B.....



比较

各自优缺点：
递归名字解析缺点：要求每台名字服务器具有较高的性能。
递归名字解析优点：1. 递归名字解析过程中，各名字服务器解析的缓存结果使用更为高效。2. 如果主机与服务器距离很远，那么采用递归名字解析将更为高效。
迭代的优缺点与上面相反。

3、目录服务操作基本概念

目录服务：除了根据实体名查找属性，用户可以根据属性描述查找实体，而不用完整的实体名

X.500： 是一组国际标准，定义了用于分布式目录服务的协议和数据模型

(1)目录结构

- 目录信息树DIT
- 目录项

  目录项是一个命名对象的信息集合。每个命名对象包括若干个属性，每个属性有一个属性类型和对应的一个或多个属性值

(2)目录服务组件

- 目录信息库DIB：DIT的数据库存储，多副本的分布式结构
- 目录系统代理DSA：类似DNS域名服务器
- 目录用户代理DUA：DUA是目录服务的用户接口，它与DSA通信
- 目录管理域DMD：DMD史背一个组织提供雨维护的DSA和DUA的集合

(3)目录服务操作

- 目录查询（响应）

  为客户提供目录查询服务，向客户提供目录信息

  响应：

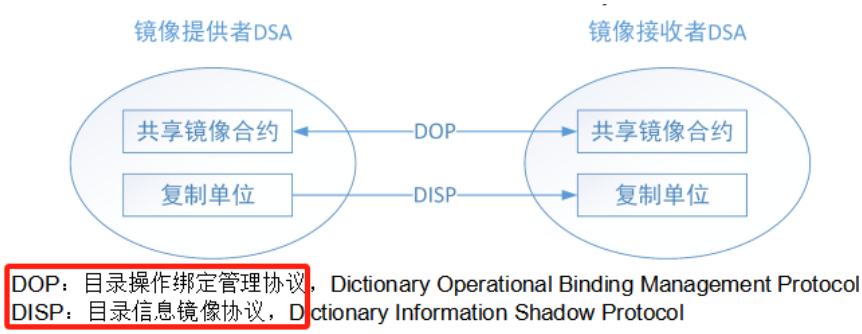
  > 成功，返回所需信息
  >
  > 失败，返回失败信息
  >
  > 转交，返回一个更适合的DSA

- 镜像操作绑定

  设置多副本，提高目录服务的性能与可用性

  镜像操作绑定（Shadow Operational Binding，SOB）管理两个正在复制全部或部分命名上下文的DSA之间的关联

  镜像操作是X.500目录服务实现多副本的手段

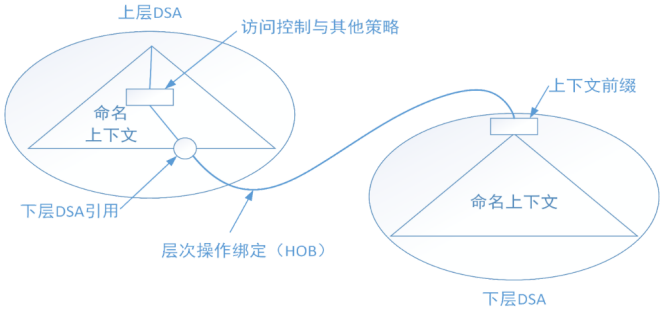


- 层次操作绑定

  在DSA之间建立层次关系，它类似于域名服务DNS区域间的"粘贴"作用

  目录信息树在DSA之间被划分为不重叠的子树

  层次绑定操作（Hierarchical Operational Binding，HOB）将这些子树的DSA按目录信息树的层次关系绑定在一起



(4)目录协议

- 目录访问协议DAP：DUA和DSA通信的协议
- 目录系统协议DSP：是两个DSA之间的操作协议
- 目录信息镜像协议DISP：DSA用来将信息从镜像提供者传送给镜像使用者
- 目录操作绑定管理协议DOP：DSA用来层次操作绑定管理和镜像管理
- LDAP协议
  - 轻量目录访问协议LDAP(Lightweight Directory Access Protocol)
  - 最初是为了简化DAP协议访问X.500目录而产生的
  - LDAP简化了操作集
  - 相比于DAP，LDAP适用于广域网和互联网环境

4、线程与多线程、分布式进程创建、进程迁移

(1) 线程与多线程

进程

操作系统中独立存在的实体，拥有自己独立的资源

线程

线程是包含在进程中的一种实体，线程是CPU资源分配的单位，进程是其它资源分配的单位

多线程系统的实现

- 用户级线程（ULT）
  - 应用程序建立，应用程序负责线程调度和管理
  - 操作系统不知道用户级线程的存在
  - 线程切换效率高，但一个线程阻塞导致整个进程阻塞
- 内核级线程（KLT）
  - OS创建线程，OS负责线程调度和管理
  - 线程调度的系统代价大
- 轻量级进程（LWP）
  - 每个进程可含多个LWP ，LWP由内核支持
  - 用户负责创建用户级线程，随后分配每个用户级线程到一个LWP，由用户负责分配

(2)分布式进程的创建

分布式进程

分布式进程是指在分布式计算环境中运行的进程，这些进程分布在多个计算节点上，并通过网络进行通信和协同工作

分布式进程的创建

- 目标主机的选择
  - 位置策略
  - 传输策略
- 执行环境的建立
  - 为新进程建立一个执行环境，主要是地址空间
  - 父进程地址空间的各个区域被其创建的子进程所继承

(3)进程迁移

3、进程迁移

进程迁移概念

进程迁移是将一个正在运行的进程挂起，它的状态从源处理机节点转移到目标处理机节点，并在目标处理机上恢复该进程运行

高灵活性但运行开销大

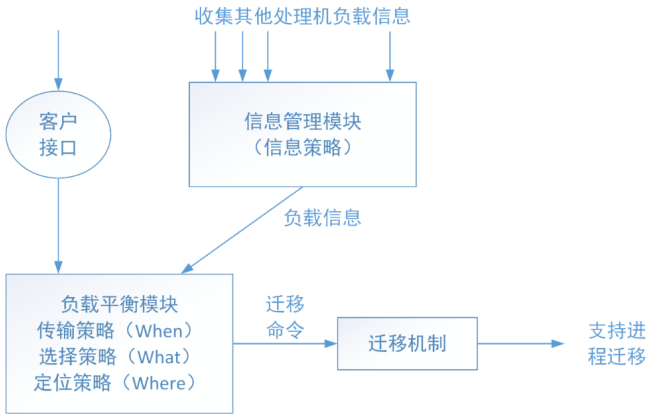
进程迁移被分解成两部分的工作：状态传输、通信转移

进程迁移的步骤

- 迁移协商，询问目标处理机是否可以接受迁移进程；
- 创建恢复进程，得到目标处理机的肯定答复后，在目标处理机上创建恢复进程；
- 中断迁移进程的运行；
- 在源处理机上收集迁移进程状态；
- 将迁移进程状态传输到目标处理机；
- 恢复被迁移状态，目标处理机上的恢复进程负责恢复迁移进程状态，重建进程实例；
- 通知被迁移进程的新位置，通知系统内其它进程迁移进程的新位置，并重建迁移中断前的通信连接；
- 迁移进程在目标节点恢复运行；
- 操作转发，利用转发机制（或利用单一系统映像的性质）保证进程可以在远程处理机透明执行。

动态负载平衡

动态负载平衡，系统中重负载处理机转移一部分负载到轻负载的处理机上运行，使得整个集群系统中的所有处理机的负载趋向均衡，从而提高系统的整体运行效率。



- 信息管理模块：衡量和收集负载信息
  - 负载信息衡量：CPU利用率、运行进程个数、资源利用率、各种资源队列的线性组合、平均队列长度、空闲主存大小
  - 信息收集策略 ： 周期性、基于事件触发；集中式、分布式
- 负载平衡模块：依据负载信息做出迁移决定

进程迁移的实现

- 状态收集
  - 内部状态收集：进程主动
  - 外部状态收集：进程挂起，操作系统执行
  - 触发式状态收集：由信号触发导致进程对自身进行状态收集
- 转发机制

  一些资源无法迁移到目的处理机（如文件管理资源），因此在进程迁移后，对这些资源的操作只能转发到原始处理机上执行

  VDPC转发机制：集群系统，北航实验室的一个项目
- 通信恢复
  - 被迁进程的新地址识别
    - 建立进程地址映射表

      在用户进程发送/接收消息时，先进行地址匹配，引导以旧地址为目的地的消息转向新的地址
    - 采用特殊的路由方式

      在用户进程发送/接收消息时，先进行地址匹配，引导以旧地址为目的地的消息转向新的地址
  - 保证不丢失任何消息
    - 消息驱赶方法

      将所有中途消息驱赶到目标进程之后，再进行进程迁移过程，并保证在迁移过程中不再发送任何“中途消息”
    - 消息转发方法

      进程迁移后，被迁移进程旧实例不中断，中途消息仍然发送到旧地址，由被迁移进程旧实例将消息转发到被迁移进程的新地址。

    - 维护消息正确顺序
        - 长消息分片，确保分片的顺序
            - 增加标志信息
            - 原子通信
        - 不同消息的先后顺序
            - 消息附加消息序号
            - 采用特定机制保证处于迁移临界区消息的正确顺序
- 进程迁移算法
    - 异步迁移算法

      这类算法允许非迁移进程在迁移过程中继续运算，只有迁移进程被中断进行相关的操作
    - 同步迁移算法

      这类算法在迁移过程中所有进程（包括非迁移的协同进程）都被挂起，进程之间需要同步来清空通信信道中的中途消息，所有进程均要阻塞等待迁移事件完成后，才能从中断处继续运行
    - 类异步迁移算法

      类异步迁移算法中尽管非迁移进程也像在同步算法中那样被中断，但是它允许非迁移进程在迁移过程中继续计算，只是在迁移过程的某些时刻进行简单的协调工作

5、WSDL的基本概念与支持的操作类型

（略，在问题15展开讲）

6、区块链共识协议、区块链关键密码学技术（Merkle树）

(1)区块链概述

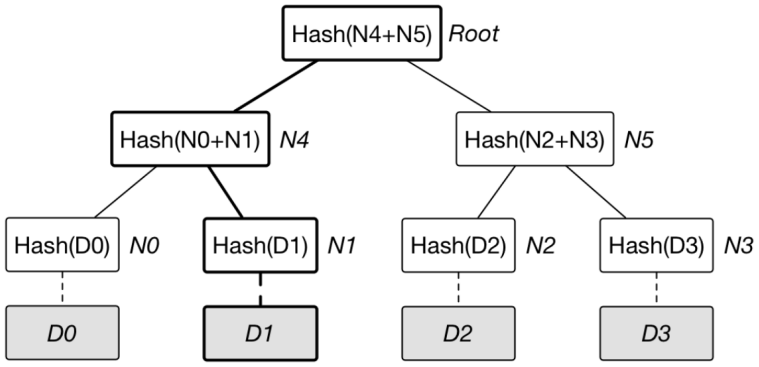
区块链（Blockchain）是一种分布式数据库技术，以链式数据块的形式存储数据，每个块都包含了前一个块的信息，形成了一个不断增长的链。这个链式结构通过加密技术确保了数据的安全性和不可篡改性。

(2)区块链密码学

哈希函数

- 哈希函数用于将任意长的消息M映射为较短的、固定长度的值，记为H(M)
- 密码角度看，哈希函数是一种单向密码体制，即从一个明文到密文是不可逆映射

Merkle树（默克尔树）



- 以上图为例，基于数据D0……D3 构造默克尔树，如果D1 中数据被修改，会影响到N1，N4 和Root

- 一旦发现某个节点如Root 的数值发生变化，沿着Root --> N4 --> N1，最多通过O(logN) 时间即可快速定位到实际发生改变的数据块D1

安全计算

(2)区块链共识算法

区块链共识机制是一整套由协议、激励和想法构成的体系，使得整个网络的节点能够就区块链状态达成一致

- 非拜占庭错误：出现故障，但不会伪造信息
- 拜占庭错误：伪造信息恶意响应的情况
- 共识算法
    - 针对拜占庭错误的，往往容错性较高，但系统性能相对较差（PoW、PoS、PBFT、DPoS等）
    - 针对非拜占庭错误的，性能较高，但容错性较差、（Paxos、Raft等）

PoW工作量证明算法

工作量证明（Proof of Work）是一种基础性算法，它为矿工在工作量证明区块链上进行的工作设置难度和规则，“挖矿”是指向比特币网络添加有效区块

- 优点
    - 去中心化，将记账权公平分派到其它节点
    - 安全性高，51%攻击破坏系统共识
- 缺点
    - 挖矿造成大量资源与电力浪费
    - 网络性能较低，一个区块的生成与共识达成周期较长
    - PoW共识算法算力集中化，算力高的大型矿池是主人

Proof of Stake权益证明

lPoS权益证明要求证明人提供一定数量加密货币的所有权即可，它将PoW的算力改为系统权益，拥有权益越大则成为下一个记账人的概率越大，不像PoW耗费资源

- 持有越多，获胜概率越大

7、欺负算法的概念与工作流程

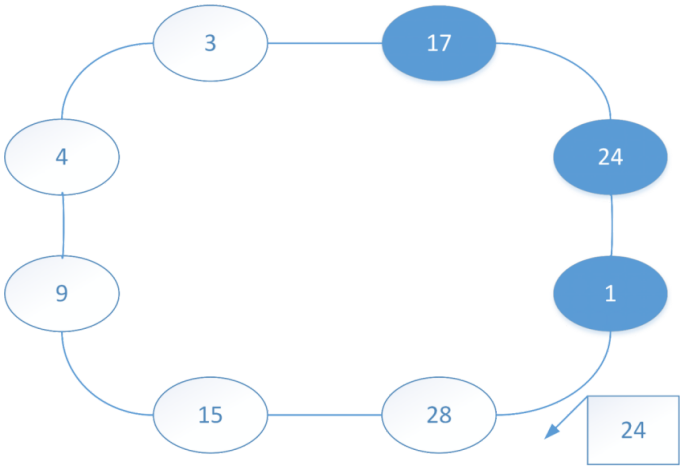
欺负算法和换算法都属于选举算法

许多分布式算法需要一个进程充当特定的角色，可采用选举算法

(1)操作步骤

举手，比大小，给下一个人（顺时针）

这种方法选出标识符最高的作为协调者



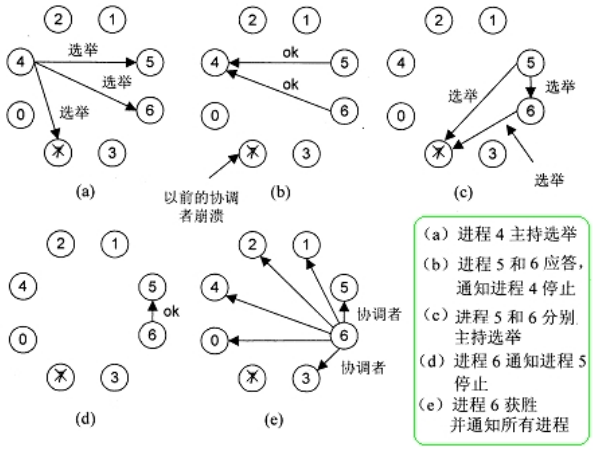
(2)复杂度分析

- 最坏情况
  - 最高编号的进程是启动选举进程的逆时针邻居
  - 选举消息转发2n+1次　（选出后）协调者消息转发n次
  - 代价：转发3n-1次消息

8、环算法的概念与工作流程

(1)操作步骤

- P向标识符更高的进程发送选举消息E(lection)
- 如果没有响应，P选举获胜
- 如果有进程Q响应应答消息OK，则P结束，Q接管选举并继续下去，直到没有比Q标识符更高的节点
- 选举获胜者向其他进程发送协调者消息C(orrdinator)



9、云计算的关键技术与核心服务

(1)云计算的概念

云计算是一种模型，人们可以使用它方便地按需通过网络访问一个可配置的计算资源，如网络、服务器、存储、应用和服务器等的共享池，只需最小化的管理工作量或服务提供商干预就可以快速地开通和释放资源

(2)云计算的关键技术

- 虚拟化技术（最常用的关键技术）
  - 分：将一个物理机器虚拟化成多个逻辑机器
  - 合：将若干分散的物理机器虚拟化为一个大逻辑机器
  - 以实现更高的利用率、资源整合、节约成本
- 一些云平台的技术
  - Google云技术（MapReduce、GFS、BigTable、Chubby）
  - IBM“蓝云”计算平台
  - Amazon的弹性计算云
  - Windows Azure
  - 阿里云
  - 华为云

(3)云计算的核心服务

云计算的核心服务，通常实现下面三类服务之一：

	服务内容	服务对象	使用方式	关键技术	系统实例
IaaS	提供基础设施部署服务	需要硬件资源的用户	使用者上传数据、代码、环境配置	数据中心管理技术、虚拟化技术等	Amazon EC2、Eucalyptus等
PaaS	提供应用程序部署与管理服务	程序开发者	使用者上传数据、程序代	海量数据处理技术、资源管理与调度技术	Google App Engine、Microsoft Azure等
SaaS	提供基于互联网的特定应用程序服务	企业和需要软件应用的用户	使用者上传数据	Web服务技术、互联网应用开发技术等	Google Apps等

10、法定多数表决复制写协议的约束条件，正确读写集团的要求

(1)基本概念

复制写协议是一种一致性协议

复制写协议中，写操作同时再多个副本上执行

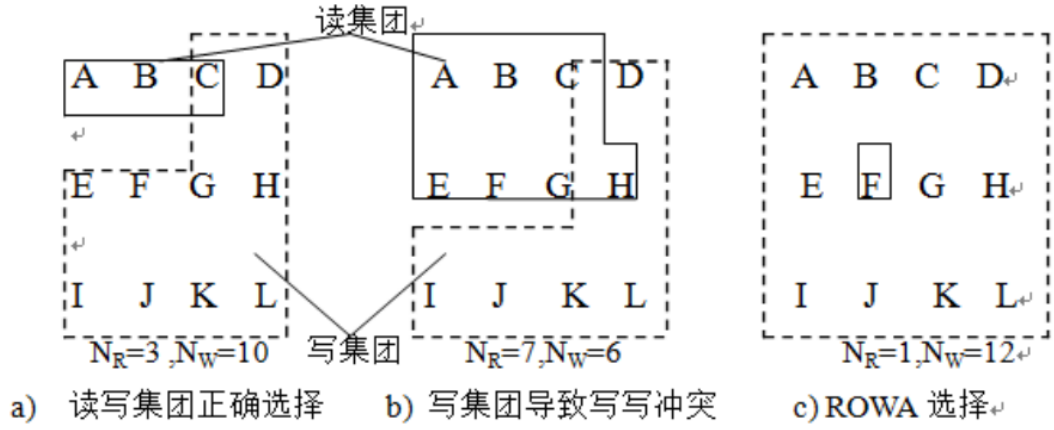
- 主动复制：写操作在所有副本上执行
- 基于法定数量的协议：写操作在法定数量的副本上进行
  - 基本思想：

    > 要求客户在读或写一个多副本共享数据项之前，向多个副本服务器提出请求，并得到它们的同意。

(2)法定多数表决复制写协议

- Thomas提出的
  - N份副本
  - 要更新一个文件，客户至少取得N/2 + 1个服务器的同意
  - 要读一个文件，客户也至少联系N/2 + 1个服务器，请求版本号
- Gifford提出的
  - 读集团N_r　写集团N_w

- $N_r + N_w > N$ 防止读-写冲突
- $N_w > N/2$ 防止写-写冲突



ROWA：读一个，写全部

11、SOAP协议的概念、基本SOAP消息的元素与结构

（略，在问题15展开讲）

12、一致性模型，数据为中心的一致性模型、客户为中心的一致性模型

(1)基本概念

一致性模型是数据存储与访问数据存储的进程之间的一种契约，遵守契约，则数据存储正确工作。

不一致的产生：

- 数据“陈旧”（暂时）
- 不同副本上使用了不同的操作定序
- 并发访问造成冲突：读写冲突（一个写，多个读）；写写冲突（多个写），此时需要一个全局定序

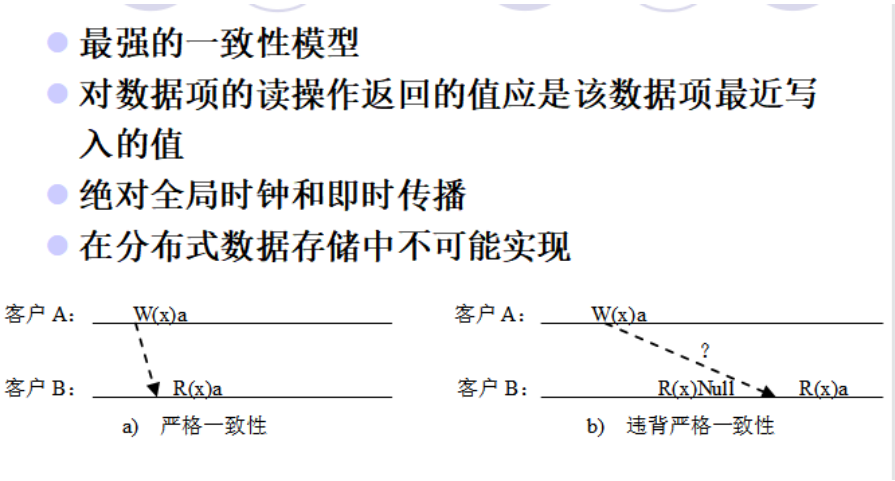
(2)数据为中心的一致性模型

从数据更新的角度，保证一致性

*实线表示客户进程对数据存储副本执行操作 *

虚线表示写操作向其它副本传播修改

- 严格一致性

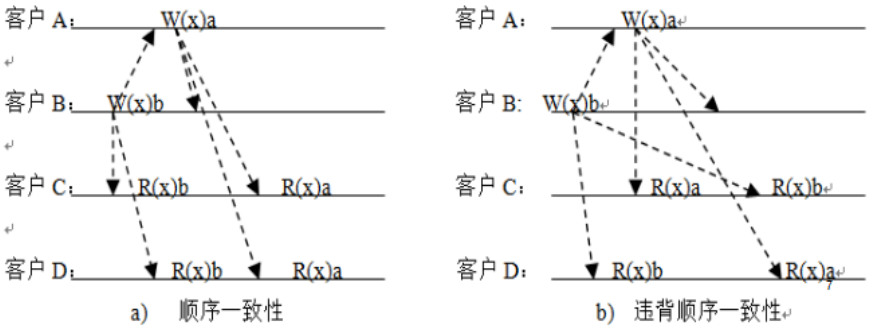


> a)客户A对x执行写操作，写入a，随后写操作传播到B，B读x的值，结果为a
>
> b)客户A对x执行写操作，写入a，在写操作传播到B之前，B读x的值，结果为null，违背一致性

只有一个有效的全局定序

- 顺序一致性

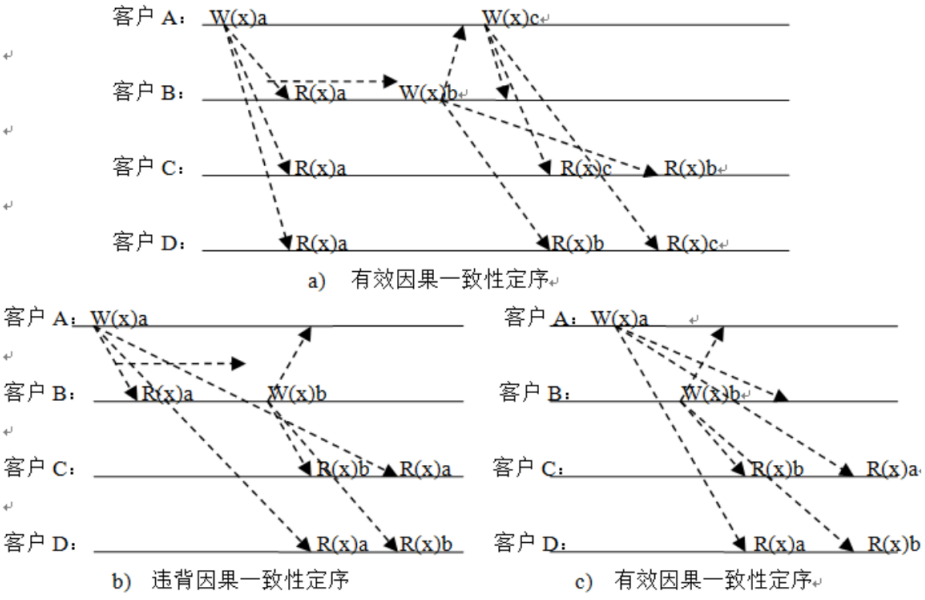
  - 放弃了时间定序的要求
  - 任何对数据存储的一组操作执行结果是相同
  - 所有客户以同样的次序看到所有写操作的全局定序



> (a)客户C和D都是先看到x写b，再看到x写a
>
> (b)略

存在多个有效的全局定序

- 因果一致性
  - 如果前个写操作可能影响后一个写操作的值，这两个写操作之间可能存在因果关系
  - 因果一致性弱化了顺序一致性，只要求因果关系的写操作在所有的副本上看到按同样的次序被执行
  - 如果两个写操作没有因果关系，说明它们是并发的
  - 并发写操作在各个副本上可以按任何次序执行，只需要遵循程序规定的次序

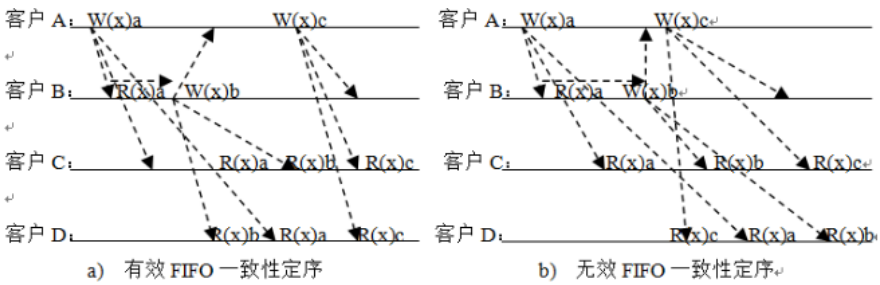


> 两个w操作之间有虚线：存在因果关系
>
> 在(a)(b)中，W(x)a和W(x)b存因果关系，因此客户C和D应该看到这两个写操作的相同定序，即先R(x)a再R(x)b

> 在(c)中，W(x)a和W(x)b是并发的，各个副本看到的写操作可以按照任意次序

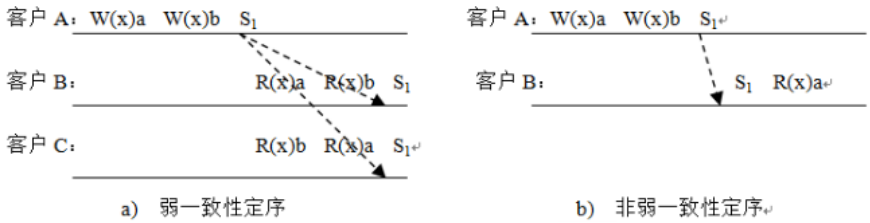
- FIFO一致性

  如果在一个节点上发生的写操作 A 在时间上早于另一个写操作 B，那么在系统的所有节点上，A 的效果必须在 B 的效果之前得以体现

> - **一个客户的写操作定序在所有副本上是相同的**
> - **也称为管道RAM模型，B、C、D都看到A先执行W(x)a，再执行W(x)c**
>
> 
>

> (a)客户A有两个写操作Wx(a)和W(x)b，C和D看到这两个写操作的顺序相同，复合FIFO一致性
>
> W(x)a和W(x)b之间存在因果关系，但是C和D看到这两个写操作的顺序不同，不符合因果一致性

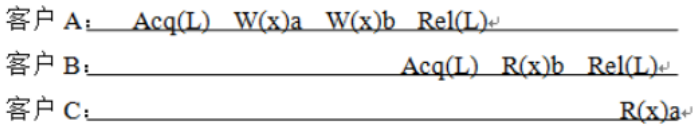
- 弱一致性

> - **采用按一个操作组，而不是单个操作进行一致性定序，通过同步变量S，来同步数据存储的所有副本**
>   1. 对数据存储相关联的同步变量的访问顺序是一致的
>   2. 每个副本在完成所有先前执行的写操作之前，不允许对同步变量进行操作
>   3. 所有先前对同步变量执行的操作在完成之前，不允许对相关数据项进行任何读操作或写操作
>
> 
>

> (a)B和C两次读取x都发生在同步操作S1之前，因此B和C看到的东西无法保证)
>
> (b)B的读操作发生在同步操作之后，它看到应该是b而不是a

- 释放一致性
  - 在弱一致性模型里，对同步变量只有一种操作synch，数据存储不能区分是已结束写的同步操作，还是读操作开始前的同步操作
  - 临界区（critical region）里互斥算法可用来解决这个问题，将同步变量的一种操作synch变成进入临界区和退出临界区两种同步变量操作

- 获取操作（Acquire）
- 释放操作（Release）

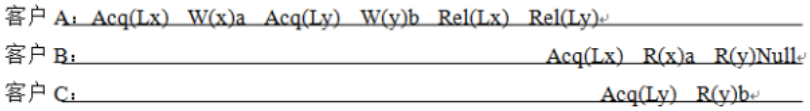


- 积极释放一致性（正常释放）：释放后将修改后的数据发给应该拥有该数据副本的其他进程
- 懒惰释放一致性（变种）：不发送，需要时，获取最新数据。

> 客户C在读取x之前没有进行获取操作，因此返回（不同步的）a值也是可以的

- 入口一致性

- 数据项一次操作与同步变量相关联

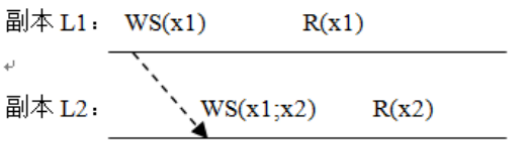


> B在读Ly之前没有进行响应的获取操作，因此读出的y是null

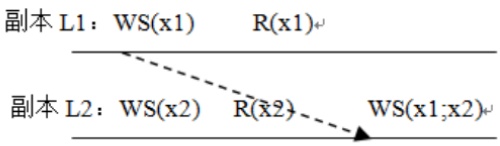
(3)客户为中心的一致性模型

从用户访问的角度，保证一致性，即只考虑客户访问的是一致的，不管数据实际上有没有更新

- 单调读
  - 如果一个进程读数据项x的值，该进程的任何后续对x的读操作总是返回前一次读同样的值或更加新的值
  - 即一个进程已经在某个副本上看到了x的值，那么它以后不可能在任何一个副本上看到x更旧的值



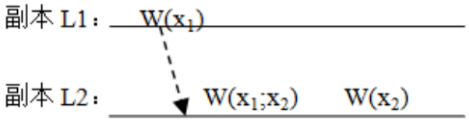
a) 单调读一致性



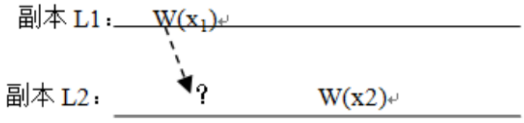
b) 非单调读一致性

> 在一个副本上的一组写操作集合标记为WS(xi)，在另一副本上执行完毕，标记为WS(xi;xj)
>
> (b)中，副本L1更新写以后，读x1（x1是最新的值），再在副本L2上读x1,由于写操作延迟传播到L2，此时的x2是旧的值

- 单调写
  - 一个进程对数据项x执行写操作，必须在该进程对x执行任何后续写操作之前完成
  - 只有所有副本都完成上一个写操作之后，才能执行新的写操作



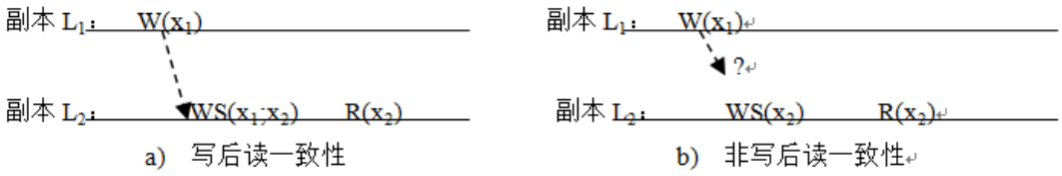
a) 单调写一致性



b) 非单调写一致性

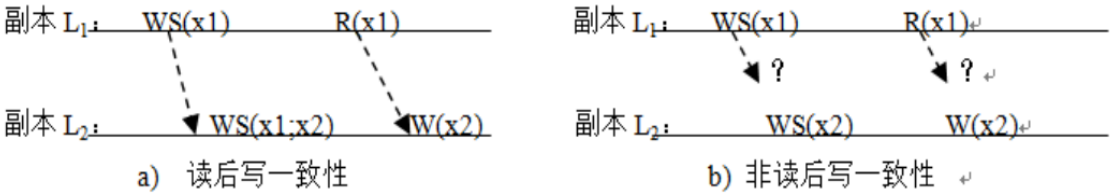
> (b)中，进程L2在副本L2上执行W(x2)之前，进程在L1上执行的写操作还没有传递到L2

- 写后读
  - 一个进程对数据项x执行一次写操作的结果，总是会被该进程对数据项x的后续读操作所看见，无论读操作发生在哪个副本上
  - 所有副本上的写操作总是在读操作之前完成



> (b)中副本L2上的R(x2)之前，副本L1上的写操作W(x1)还没有传递到副本L2，故*所有副本上的写操作*没有完成

- 读后写
  - 一个进程对数据项x的写操作是跟在同一进程对x读操作之后，保证相同的或更加新的x的值能被看见。
  - 进程对数据项x所执行的任何后续写操作总是在x的副本上执行，而该副本是用该进程最近读取的值所更新。



13、RPC、Java RMI的概念，RMI和RPC异同对比

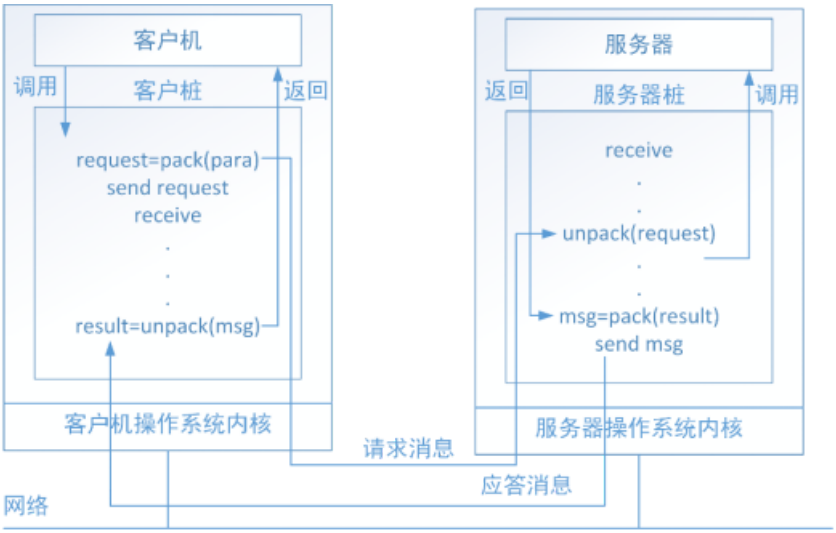
(1)远程过程调用（RPC）

RPC的概念

- 为了解决隐藏通信而产生
- 允许一台机器上的过程调用其它机器上的过程
- 调用方法将调用参数通过消息传送给被调用方，被调方将执行结果通过消息返回给调用方。
- 这种方式对消息进行了包装，客户看不到这些消息的传送过程（隐藏通信）

RPC的调用过程

- client应用程序正常调用client stub(客户桩)
- client stub构造消息，通过trap进入操作系统内核
- 内核将消息发送到server的操作系统内核
- Server内核将消息交给server stub
- server stub将消息解包，用相应参数调用服务例程
- 服务例程进行计算，将结果返回server stub(服务器桩)
- server stub 构造消息，通过trap进入内核
- 内核将消息发送到client的内核
- Client内核接收消息并将它交给相应stub
- stub解包，将调用结果返回应用程序



RPC参数传递

- 值参数传递：数据格式问题
  - 统一规范格式，简单但不灵活，效率也不高
  - 增加一个参数，或在消息中指明格式，由接收者做相应处理，效率较高，但每个机器需配置完全的格式转换程序
- 指针类参数问题
  - 全部禁用，不利于用户使用
  - 复制/重新存储方法，客户桩将调用参数从堆栈读到一个缓冲区，使调用参数从层次形式变为扁平形式，在消息中传递给服务器。服务器桩解码，回复层次形式，然后调用服务进程。（结果返回是类似的过程）
- 复杂数据类型问题
  - 在GIS中，指针指向图，此时需制订专门的通信协议，详细描述数据规格，并配置相应程序读取

RPC语义

- 客户机不能定位服务器
- 请求消息丢失
- 应答消息丢失
- 服务器崩溃
- 客户崩溃

DCE（分布式计算环境） - RPC实现

(2)远程方法调用（RMI）

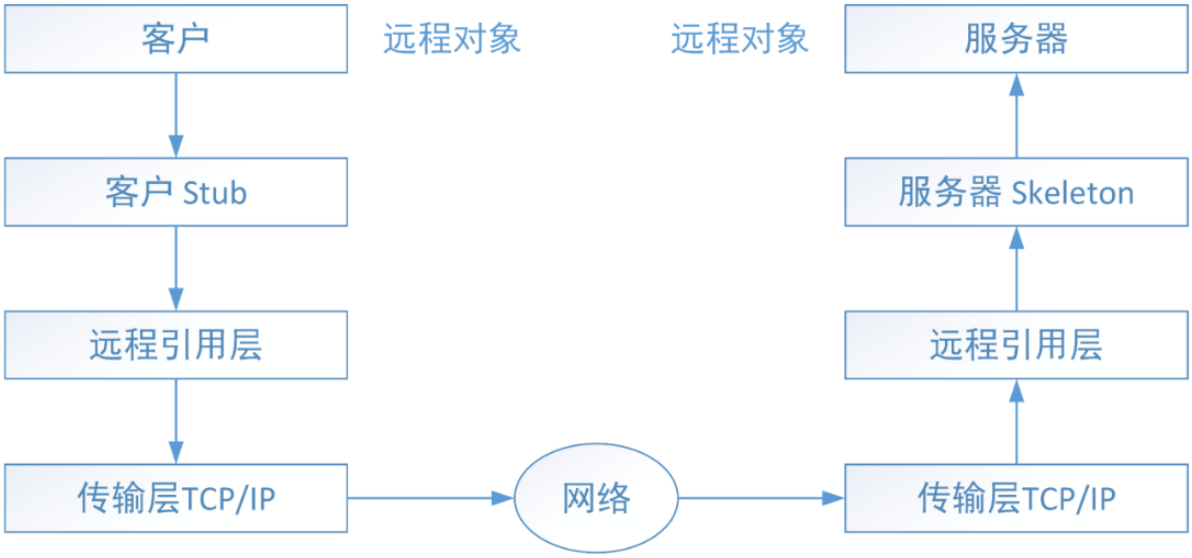
RMI的概念

远程方法调用RMI（Remote Method Invocation）是Java特有的分布式计算技术。它实质上是通过Java编程语言扩展了常规的过程调用，在网络上不仅可以传送对象的数据，而且可以传送对象的代码。

RMI体系结构：

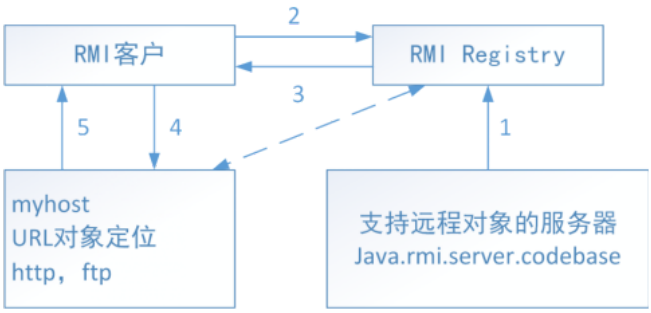
通常RMI系统由下面几个部分组成：

① 远程服务的接口定义。
② 远程服务接口的具体实现。
③ 桩(Stub)和骨架(Skeleton)文件。
④ 一个运行远程服务的服务器。
⑤ 一个RMI命名服务，它允许客户端去发现这个远程服务。
⑥ 类文件的提供者(一个HTTP或者FTP服务器)。
⑦ 一个客户端程序。



RMI的调用过程

- 远程对象注册与名字绑定
- 客户按名字查找远程对象
- 注册器返回远程对象接口
- 客户从codebase请求stub类
- http服务器返回远程对象的stub类



服务器要向RMI Registry注册服务，客户端通过RMI Registry获取服务位置

(3)RMI和RPC异同对比

- 同

  都是远程调用的概念，都支持分布式系统中的通信，在不同计算机上的进程之间进行通信

- 异
  - 方法调用区别
    - RMI允许在远程计算机上调用远程对象的方法
    - RPC调用的是远程计算机上的过程，而不是远程对象的方法。

- 传递消息限制
  - RMI可以通过远程调用传递复杂的数据结构和对象。RMI支持参数和返回值传递的是Java对象，而RPC仅支持传递基本数据类型和自定义数据结构
- 采用协议不同
  - RMI使用avaRMI协议作为通信协议，通常基于TCP协议进行传输
  - RPC并没有固定使用一种特定的协议，而是可以基于多种传输协议实现。不同的RPC框架可以选择不同的底层协议来进行通信。
- 支持语言平台
  - RMI是Java特有的远程调用机制，RPC是跨语言的

14、IDL的基本概念，在分布式系统中的作用

(1)基本概念

IDL是一个描述软件组件接口的语言规范。 IDL用中立语言的方式进行描述，能使软件组件（不同语言编写的）间相互通信。

(2)IDL在分布式系统中的作用

- 实现标准的对象接口，构造分布式对象应用，使客户程序能调用远程服务器上对象的方法
- IDL为分布式对象系统定义模块、接口、类型、属性和方法提供了设施

15、Web服务的概念，Web服务描述语言（SOAP、UDDI）

(1)Web服务

(2)Web服务契约

概念

契约是供求双方间进行交换的一种约定，在面向服务的分布式计算系统中，契约是系统之间交换数据时应遵守的约定（技术契约），技术契约是由相应的接口执行的。

说明的问题：服务功能描述（What） 服务访问描述（How） 服务位置描述（Where）

抽象描述：

- 服务功能描述：表达了契约公开的接口。包括：
  - 端口类型（接口）定义（portType）
  - 操作定义（Operation Definition）
  - 消息定义（Message Definition）
  - 类型定义（Type Definition）
  - 策略定义（Policy Definition）

具体描述：

- 服务访问描述：为抽象描述补充相关的实现细节，即如何访问服务。包括：
  - 端口类型（接口）绑定（Port Binding）
  - 操作绑定（Operaion Binding）
  - 消息绑定（Message Binding）
  - 策略定义
- 服务位置描述，关注从何处得到所需要的服务。包括：
  - 服务定义（Service Definition）
  - 端口定义（Port Definition）
  - 地址定义（Address Definition）
  - 策略定义

(3)SOAP、WSDL、UDDI的概念

SOAP

SOAP：即简单对象访问协议，是基于XML的一种通信协议，由HTTP协议承载，与平台和引用程序语言无关，用于表示信息交换的协议。

SOAP的作用：

- 在WEB上交换结构化的和固化的信息
- 提供了一种标准的方法，使得运行在不同的操作系统并使用不同的技术和编程语言的应用程序可以互相进行通信

消息结构：

```
<soap:Envelope> (必选)
    <soap:Header>头部信息</soap:Header>
    <soap:Body> (必选)
        消息内容……
        <soap:Fault>错误和状态信息……</soap:Fault>
    </soap:Body>
</soap:Envelope>
```

WSDL

即web服务描述语言，用来描述和定位一个Web服务，规定了服务的位置，服务提供的操作（或方法）

与XML的关系：使用 XML 编写的文档，WSDL是基于XML的。

用于描述的实体：WSDL用于描述某个网络服务

文档结构：

<definitions>
<type> 数据类型定义</type>
<message> 被交换数据类型定义</message>
<portType> 一组操作…</portType>
<binding> 协议和数据格式的规范说明…</binding>
<service>端口和地址信息.. </service>
</definitions>

- WSDL支持的操作类型

  > `<operation>` 元素用于定义一个操作

  - 单向（one-way）：仅服务端接收请求/消息

    >

  - 请求-响应（request-response）：服务端接收请求/消息，并响应相关消息

    >

  - 要求-响应（solicit-response）：服务端发送请求/消息，并等待接收一个响应

    >

  - 通知（notification）：仅服务端发送请求/消息

    >

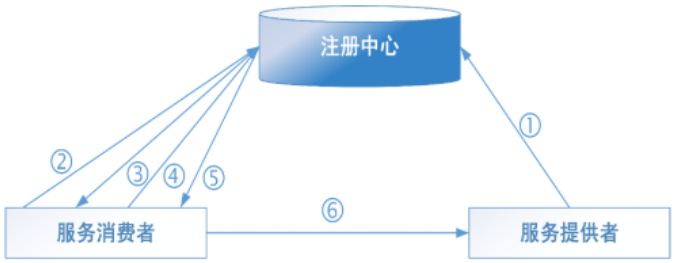
UDDI

UDDL，即通用描述、发现和集成

是Web服务提供者和消费者之间的接洽点

- 对于Web服务开发者是一个注册服务器
- 对于Web服务消费者是一个目录服务器

3、Web 服务描述、发现和集成（UDDI- Universal Description Discovery and Integration）的流程



1. 服务发布者通过 SOAP 消息，将服务描述**注册**于 UDDI。
2. 服务消费者通过 SAOP 请求消息，向 UDDI **查询**所有匹配的服务。
3. UDDI 发现所有的匹配结果，并用 SOAP **响应**消息将这些结果返回给服务消费者。
4. 服务消费者通过 SOAP **请求**消息向 UDDI 索取所选 **Web 服务**描述。
5. UDDI 用 SOAP 响应向服务消费者**返回**选定的 **Web 服务描述。**
6. 用选定的服务描述**调用选定的服务**，并集成功能更为强大的 Web 服务。

16、SOA的概念架构与服务组件架构

(1)SOA

SOA试图解决的问题

> 企业业务模式的变化：传统的业务部门的消失，如企业运输部门
>
> 过去的IT系统建设以部门为基础整合，是部门内的垂直整合；现在需要在企业各部门间进行水平整合
>
> 企业IT系统抽象程度低，与业务之间存在着断层。
>
> 企业IT系统改变或者升级时，原有的硬件和软件资源希望在新系统中尽可能重用

SOA(Service-Oriented Architecture)

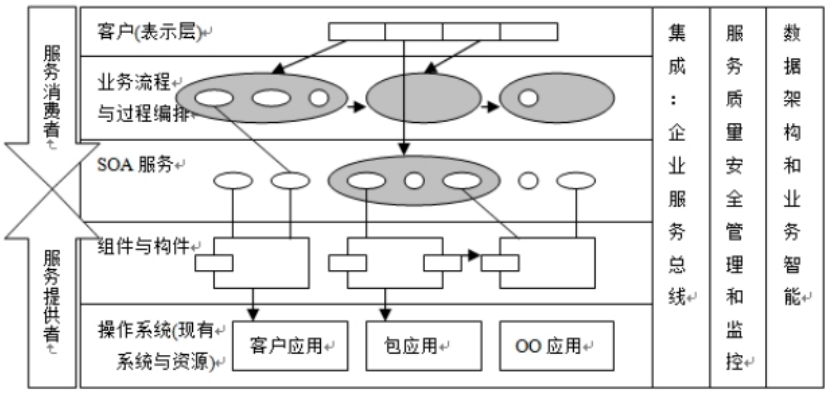
SOA即面向服务的架构，是一种软件设计和架构模式，其主要思想是将软件系统划分为可重用的服务单元，这些服务单元通过标准化的接口进行通信，从而形成一个松散耦合的系统。

每个服务单元执行特定的业务功能，并且可以独立开发、部署和维护。

SOA旨在提高系统的灵活性、可维护性和可扩展性，同时促进系统的集成和协作。

SOA概念架构

SOA概念架构：采用分层模式，这个架构自底向上是操作系统层、服务组件层、服务层、业务流程编排层和访问表现层。



(2)SCA、SDO、BPEL

SCA（Service Component Architecture，服务组件架构）

SCA是一种用于构建面向服务的架构的开放性标准。它提供了一种模型和规范，用于开发和部署服务组件，使得这些组件可以被集成到分布式应用程序中。

SCA 的目标是简化面向服务的开发，提高组件的可重用性，并促进不同类型的组件之间的互操作性。

SDO（Service Data Objects，服务数据对象）

SDO是SOA编程模式中不可或缺的业务数据模型，它可以与多种数据模式交互（关系数据库、XML数据库、Web服务等等）

BPEL（Business Process Execution Language，业务流程执行语言）

BPEL是一种业务流程定义语言。它以业务流程及其参与者的交互为基础，定义了业务流程的描述语法，用于业务流程建模。

三者关系

- SCA是服务组件架构，关注服务，用一致的方式构建和使用不同的服务
- SDO是服务数据对象，关注数据，SDO整合的数据可在BPEL流程中无障碍地流动
- SDO是服务数据对象，关注数据，SDO整合的数据可在BPEL流程中无障碍地流动

17、设计题：IDL到Java的转化

例题1

试用OMG的IDL定义一个银行储蓄个人账号接口Account和一个储蓄账户管理接口AccountManager，账号接口Account有一个获取账户余额的方法balance()，账户管理接口AccountManager有一个能打开指定账户的方法open()

```
Module BankingSystem{

    interface Account{
        readonly attribute long ID;
        attribute double myBalance;
        double balance();
    }

    interface AccountManager{
        Account open(in long ID);
    }

}
```

IDL转JAVA：

```java
package BankingSystem;

public interface AccountOperations{
    int ID();
    double myBalance();
    void myBalance(double newMyBalance);
    double balance();
}
```

```java
package BankingSystem;

public interface AccountManagerOperations{
    BankingSystem.Account open(int ID);
}
```

例题2