1.1 什么是分布式计算系统？它的实质是什么？

分布式计算系统是由**多个计算机相互连接**组成的一个系统，这些计算机**在分布式操作系统的环境下**，**合作**执行一个或多个任务，最少依赖于集中的控制过程、硬件和数据

实质：分布式计算系统=分布式**硬件**+分布式**控制**+分布式**数据**

1.2 什么是分布式系统的开放性，实现开放性的基本技术途径是什么？

开放性是指**能否用各种方法进行扩展和实现**。

基本技术途径：基于统一的**通信协议**和公开的访问共享资源的**标准化接口**

1.3 什么是分布式系统的可扩展性，可扩展性设计面临的挑战是什么？

可扩展性是指**一个系统在资源数量和用户数量增加时仍能有效工作**

挑战：控制物理资源成本、控制性能损失、防止资源耗尽、避免性能瓶颈

1.4 什么是分布式系统的异构性，如何隐藏分布式系统的异构性？

异构性：具体表现在网络、计算机硬件、操作系统、编程语言、开发商的异构性

隐藏方法：中间件、虚拟机

1.5 什么是分布式系统的透明性，过分强调分布式系统的透明性会带来什么问题?

透明性是指**向用户和应用程序隐藏了分布式计算系统部件的差异，系统被认为是一个整体**（**访问**、**位置**、并发、失效、复制、迁移、性能、规模）

问题：可能难以实现，比如位置透明性，受到节点之间距离和通信速度的限制；还要考虑系统其他要求，多副本的一致性要保证，就会导致透明性难以实现。

1.8 什么是分布式系统的安全性？分布式系统的安全性的基本技术途径是什么？

安全性是指**建立在因特网上的重要信息应该是保密的**

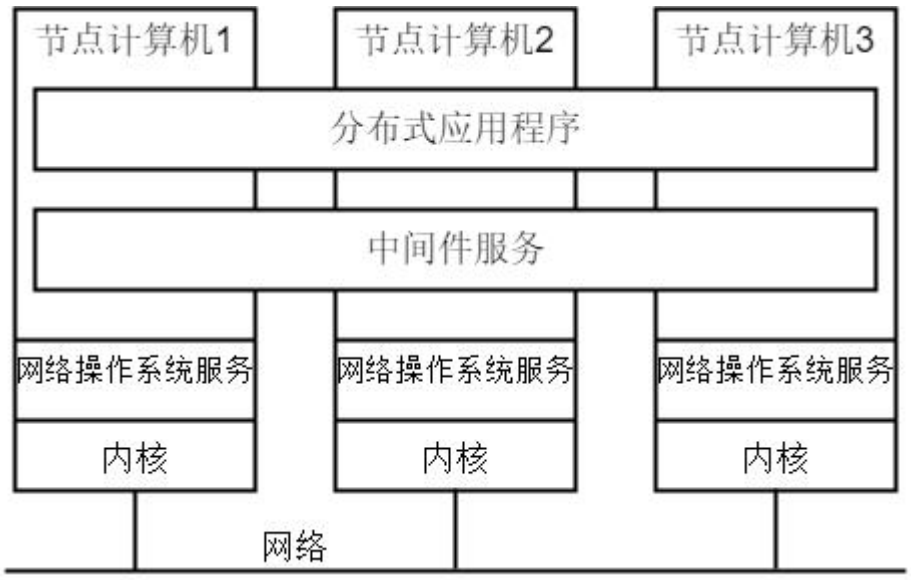
基本技术途径：身份认证、消息加密、访问控制。

1.16 什么是中间件，它的功能是什么？它在分布式系统中的地位是什么？

中间件是**一种独立的系统软件或服务程序，屏蔽了节点计算机之间的差异，为应用程序提供了统一的运行环境**。

功能：**命名服务**、**作业调度**、**高级通信服务**、**资源管理**、**数据持久性**、分布式事务、分布式文档系统、安全服务。

地位：中间件的一个重要目标是**对应用程序隐藏底层平台的异构型**，因此中间件系统都提供一组完整度不同的服务集。这些服务是通过中间件系统提供的接口来调用的。一般**禁止跳过中间件层**直接调用底层操作系统的服务。



1.18 分布式系统有哪些计算模式

面向对象模式 OOM：基于客户/服务器模型，服务器以面向对象的技术实现。

面向服务模式 SOM：基于客户/服务器模型，服务是动态的。

公用计算模式 UBM：支持e^{-}科学的计算，计算量巨大。这个模式如同公用事业，为客户提供需要的算力，免去客户对设备的采购、安装和维护。

志愿参与模式 VJM：充分利用网上空闲的计算能力，支持计算量巨大的科学计算。

2.1 实体的名称、地址、标识符和属性的区别是什么？

实体在一个计算机系统中是指范围广泛的事物。

实体的名称是一个用户可读的、便于记忆的字符串。

实体的地址是实体的访问点，一个实体可以有多个访问点。

实体的标识符是实体在系统内部的唯一标识符，可以提高访问效率。

实体的属性是描述实体特征的若干键值对<类型-值>。

2.5 有哪些名字服务形式？名字服务器的组成与功能是什么？

服务形式：

名字服务：命名实体和其属性（地址）

目录服务：命名实体与其一个或多个属性绑定的集合

合约服务：一种增强的目录服务

组成：

名字服务器操作：包括上下文管理、查询操作和行政管理

名字解析：根据请求，利用数据库的上下文信息对名字进行解析，得到对象 IP

缓存：缓存名字查询和解析的结果

多副本管理：副本修改和副本一致性维护

通信：包括与客户端的名字代理和名字服务器之间的通信

数据库操作：存放名字解析上下文或其子域

功能：上下文管理、名字查询、和其他服务的通信协调

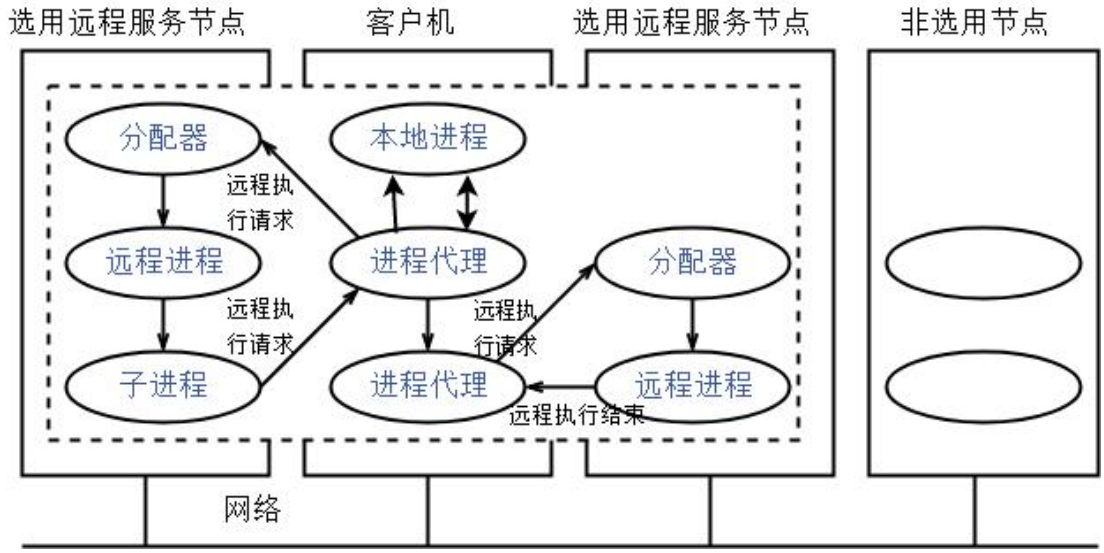
2.14 什么是目录服务？目录项和属性及属性值的关系是什么？

目录服务：一种特殊的名字服务，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匹配的实体列表。

关系：**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**。

3.7 什么是远程执行逻辑机模型？对逻辑机模型的要求是什么？

逻辑机模型：客户节点上的代理负责远程服务节点上远程进程执行的初始化；远程服务节点执行客户赋予的进程。这个模型称为逻辑及模型。



要求：

(1) 应有一种机构来传播空闲处理机的可用信息，或识别分布式系统中的这种空闲处理机**（寻找管理机制，传播空闲机信息）**

(2) 远程执行应像进程在本地执行那样容易实现**（进程远程执行要是透明的，应与位置无关）**

(3) 主人有限原则**（主人优先原则，当空闲机的拥有者要用时，应该停止远程执行，归还工作站）**

3.8 进程迁移为什么没有被当前大多数操作系统所支持？

1. 这些 OS 最初都是为了单机运行而设计的，对迁移进程适应性较差
2. 没有足够的应用需求迫使 OS 厂商支持进程迁移

3.11 如果文件 F 与机器是紧密绑定的，进程迁移后如何实现对文件 F 的调用？

通过转发机制，将对文件 F 的操作转发的原始处理机上执行。

3.12 进程迁移如何保证与其他进程的通信不被丢失？

进程迁移后通信丢失的三种问题：

1) 被迁进程到的新地址
   1. 建立进程地址映射表
   2. 采用特殊路由方式
2) 保证不丢失任何消息
   1. 消息驱赶方法
   2. 消息转发方法
3) 维护消息正确顺序（2 层含义：长消息分片顺序、不同消息先后顺序）

   长消息分片顺序：1.增加标志信息、2.原子通信

   不同消息先后顺序：1.消息附加消息序号、2.采用特定机制保证处于迁移临界区消息的正确顺序

3.13 何为异步进程迁移算法？何为同步进程迁移算法，他们的优缺点是什么？

异步迁移算法：允许进程在迁移过程中**继续执行**

优点：执行效率好

缺点：和原有环境的兼容性不好、不方便移植

同步迁移算法：所有进程在迁移过程中**被挂起**

优点：易于实现，有较好的可移植性

缺点：所有进程被迫中断，效率不高，需要中央控制管理进程参与

类同步迁移算法：所有进程被中断，但在**迁移过程中可以执行**

3.15 比较进程远程执行和进程迁移两种机制

进程远程执行是在集群中或网络中寻找一个或多个合适节点来执行用户程序

进程迁移是将一个正在运行的进程挂起，将他的状态从源处理机转到目标处理机节点，并在目标处理机上恢复该进程。

补充：负载均衡的策略/方式，负载均衡的触发方式

负载均衡策略：

传输策略：何时迁移（when）

选择策略：迁移何种进程（what）

定位策略：迁移到哪个节点上（where）

触发方式：

中央服务器触发：通过中央控制方式触发

发送者触发：节点发现自己过载，寻找负载轻的节点，触发迁移

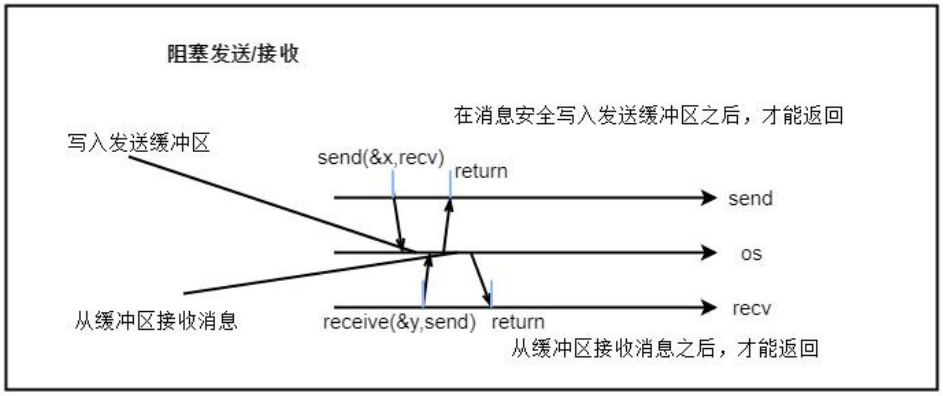
服务者触发：节点发现自己负载轻，主动找其他负载重的节点，负担负载

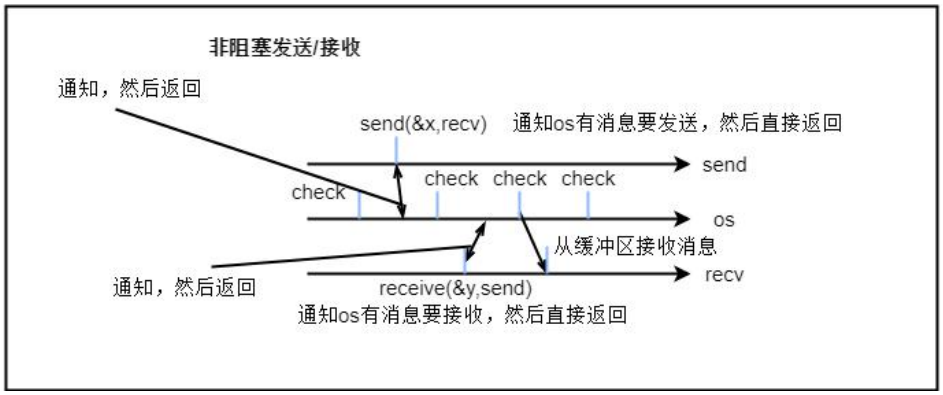
对称触发：发送者触发和服务者触发的综合

自适应触发：一种优化方法，根据系统变化，自适应地选择触发方式

4.1 在水平时间轴上表示阻塞发送/接收和非阻塞发送/接收与操作系统内核之间操作的时间关系

4.1在水平时间轴上表示阻塞发送/接收和非阻塞发送/接收进程与操作系统内核之间操作的时间关系。





4.2 试叙述如何实现阻塞发送/接收和非阻塞发送/接收，对操作系统有什么要求?

实现看书

要求:

阻塞发送/接收：操作系统提供临时缓冲区

非阻塞发送/接收：操作系统提供临时缓冲区+操作系统检查是否有发送/接收通知

4.3 如何实现“至多一次”操作语义?

至多一次语义，保证正确完成消息传送至多一次。在没有节点崩溃和网络断开的情况下，只正确执行一次消息传送。

可以通过只正确传送一次消息来实现，这样的话，最终结果就是消息没传送或者正确传送过去一次。

可以参考 Kafka 的至多一次语义，接收端在接收到消息后，在处理消息之前，自动提交反馈，使系统认为消息已经传送成功。

4.4 对以下每个应用程序，你认为“至多一次”和“至少一次”语义哪个最好?

（1）在文件服务器上读写文件：至少一次。

（2）银行服务：至多一次

（3）编译一个程序：至少一次

4.12 RPC 被认为是分布式最初的中间件，它能实现分布式的透明性吗？

在执行 RPC 过程中，客户可以像执行本地调用一样调用远程过程，并不直接执行 send 和 receive 原语，不关心消息的传递，所有这些都被隐藏在桩/骨架代理里面，从而实现 RPC 的透明性。

5.6 在集中式互斥算法中，若考虑进程的优先权，算法应该如何设计？

集中式互斥算法：选一个进程作为协调者，当一个进程要访问临界区，先向协调者发送一个请求消息，如果当前临界区没有进程访问，则返回应答消息，允许申请者访问。

考虑优先权：选一个进程作为协调者，当一个进程要访问临界区，先向协调者发送一个请求消息，如果当前临界区没有进程访问，则返回应答消息，允许申请者访问；如果当前临界区有进程访问，则将申请者加入一个优先级队列，等当前访问进程结束，取出优先权最大的申请者，返回应答消息，允许访问。

5.7 Ricart_Agrawala 算法如何改进了 Lamport 算法，它的优点是什么？

Lamport 需要发送 $3(N-1)$个消息，而 Ricart 只要发送 $2(N-1)$个消息

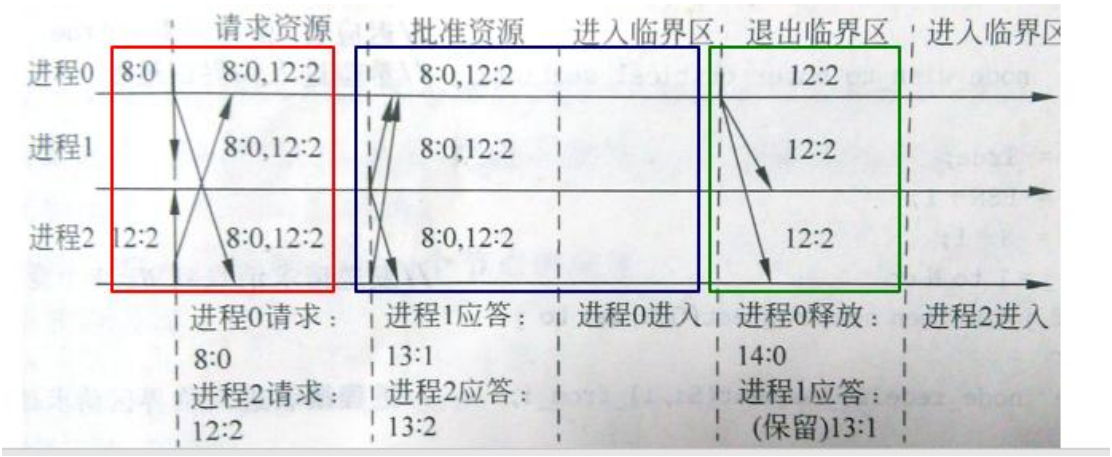
优点：具有对称性，具有完全的分布式控制，对通信链路相对速度具有不敏感性

Lamport 算法：

1. 进程P_i发送资源请求消息$Request(T_i: P_i)$到其他进程（包括自己）的请求队列中
2. 当P_j收到 $Request(T_i: P_i)$，按时间戳顺序置入请求队列，当P_j没有资源请求或者P_j的请求晚于P_i的请求，返回给发送进程应答消息$Reply(T_j: P_j)$
3. P_i允许进入临界区的两个条件：

   消息队列中有$Request(T_i: P_i)$，且T_i 最小

   P_i已从其他进程接收到晚于T_i的消息
4. P_i退出临界区，删除自己请求队列中的$Request(T_i: P_i)$，发送给其他进程释放消息$Release(T_j + 1: P_i)$
5. 当P_j收到释放消息后，删除对应的请求消息$Request(T_i: P_i)$，检查是否还有进程等待进入临界区



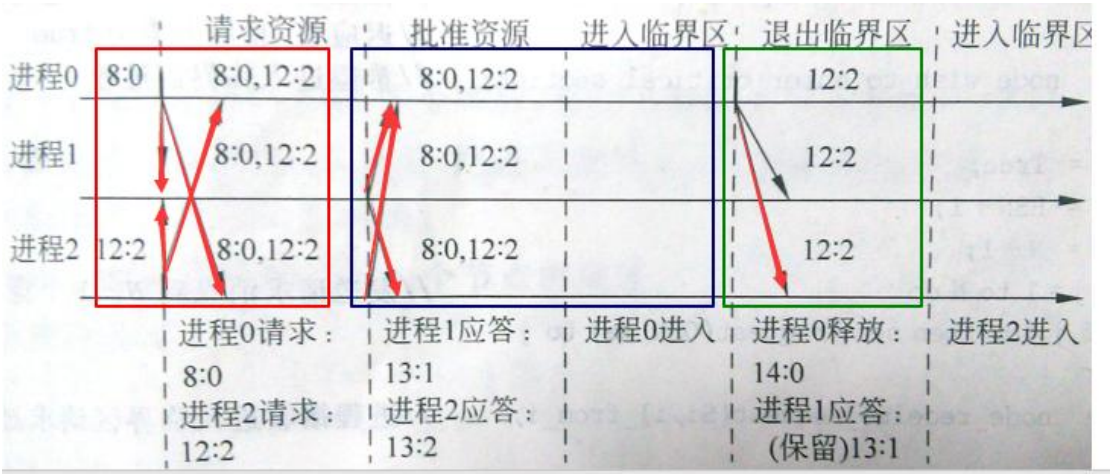
Ricart 算法：

1. 进程P_i发送资源请求消息$Request(T_i: P_i)$到其他进程（包括自己）的请求队列中
2. 当P_j收到 $Request(T_i: P_i)$，按时间戳顺序置入请求队列，并做：

如果该进程P_j没有资源请求或者P_j的请求晚于P_i的请求时，返回给发送进程应答消息$Reply(T_j: P_j)$，否则推迟返回。

资源从临界区退出，向有关的资源请求进程补发一个应答消息

3. 请求进程从其他所有竞争进程得到应答消息$Reply(T_j: P_j)$，便可进入临界区



（红线的画法是 Ricart 算法）

5.8 比较集中式算法、Ricart 算法和令牌算法的开销和问题

开销：集中式算法的开销最大，Ricart 算法的开销是 $2(N-1)$，令牌算法的开销是$N-1$。

问题：

集中式算法：容易造成单点故障，有性能瓶颈

Ricart 算法：由于不应答被认为是资源被占用，所以如果某个节点故障，会导致算法异常终止。同时各进程对资源使用情况缺乏了解。

令牌算法：检测令牌丢失问题困难

令牌算法：只有拥有令牌的进程才能进入临界区，访问资源。

逻辑环结构：

静态建立逻辑环，每个进程必须知道它的逻辑前驱地址 或 逻辑后继地址。

以后继地址为例：

进程接收一个带地址标识$addr$的令牌后，检查该地址是否为P_i的地址，如果$addr = i$，检查进程是否要进入临界区，如果是，则进入临界区，退出临界区后将$addr$置为进程的后继。路由算法得到下一个节点的地址；如果$addr! = i$，路由算法得到下一个节点的地址。

无环结构：持有令牌的进程可以把令牌发送给其他任何进程

5.11 共享 K 个相同资源的互斥算法和 Ricart 算法的共同点和区别是什么？

共同点：基于相同的概念，每个竞争进程都维护一个推迟应答数组 RD[]，数组元素是表示相应进程是否推迟发出应答消息。

区别：应答消息到达的环境。在 Ricart 算法中，正在等待进入临界区的进程要得到 N-1 个应答消息，在共享 K 个相同资源的互斥算法中，要等待 N-K 个应答消息。在 Ricart 算法中，推迟应答数组 RD[]是布尔型，在共享 K 个相同资源的互斥算法中，推迟应答数组 RD[]是整型。

共享 K 个相同资源的互斥算法：由于一种共享资源可能有多个副本（K 个），该算法允许 K 个进程同时进入临界区。N-1 个竞争进程中，少于 K-1 个进程在临界区中，则另一个进程可以进入临界区，但是需要得到$N-1-(K-1)\ =\ N-K$个应答消息

基于事件优先权算法：本算法适应优先权（进程优先权，P 系统）系统和实时（进程执行残留时间，RT 系统）系统

完全可靠网络算法：

每个节点上存在两个请求队列 P 和 Q:

P：放置其他节点来不及处理的随令牌转来的请求

Q：放置其他节点送来的请求

1 进程 i 希望进入临界区，发送$Request(i,P(i))$到其他所以竞争进程，并将$(i,P(i))$存入接收进程的 Q 队列，队列按优先级>>排序;

2 拥有令牌的进程 j 退出临界区，检查本地的队列 Q 和随令牌同时被接收的队列 P:

如果 P 和 Q 都空，表明没有进程要进入临界区，j 正常工作

如果 P 空，Q 不空，则在 Q 中标注最高优先权的进程，合并队列形成新的 P 队列

如果 P 不空，Q 空，则在 P 中标注最高优先权的进程，合并队列形成新的 P 队列

如果都不空，则在 Q 中标注最高优先权的进程，合并队列 PQ 形成新的 P 队列

3 进程 j 将令牌和新的队列 P 发送给所标注的最高优先权的进程

不可靠网络算法：

实际情况：节点（进程）可能失效；消息丢失或不按顺序发送（5.12）

算法过程和可靠网络算法基本一致，但需要一个超时机制：

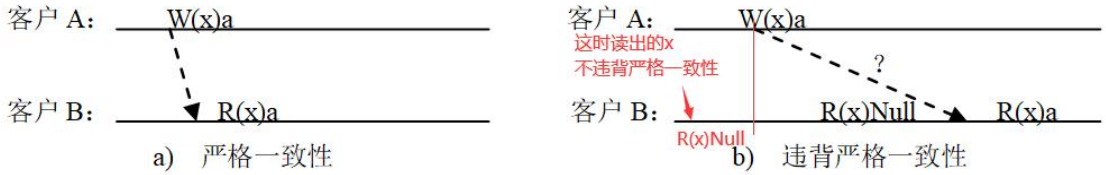
当进程 j 在发送令牌到另一个进程 i 时，启动超时机制

如果进程 j 在超时范围内没有收到 i 的应道消息，则认为 i 失效，修改队列 P（删除进程 i），发送一个令牌和新的队列 P 给下一个进程

8.2 图 8.1（b）为什么违背严格一致性？

严格一致性：对数据项的读操作返回的值应是该数据项最近写入的值

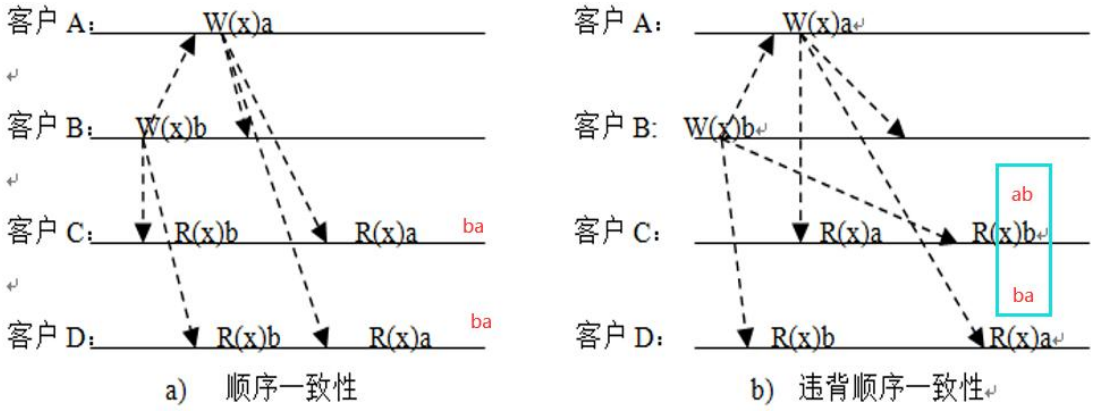
原因：B 读 x 读到的不是 a，而是 Null



8.3 图 8.2（b）为什么违背顺序一致性？

顺序一致性：所有客户看到的是同一个有效全局定序

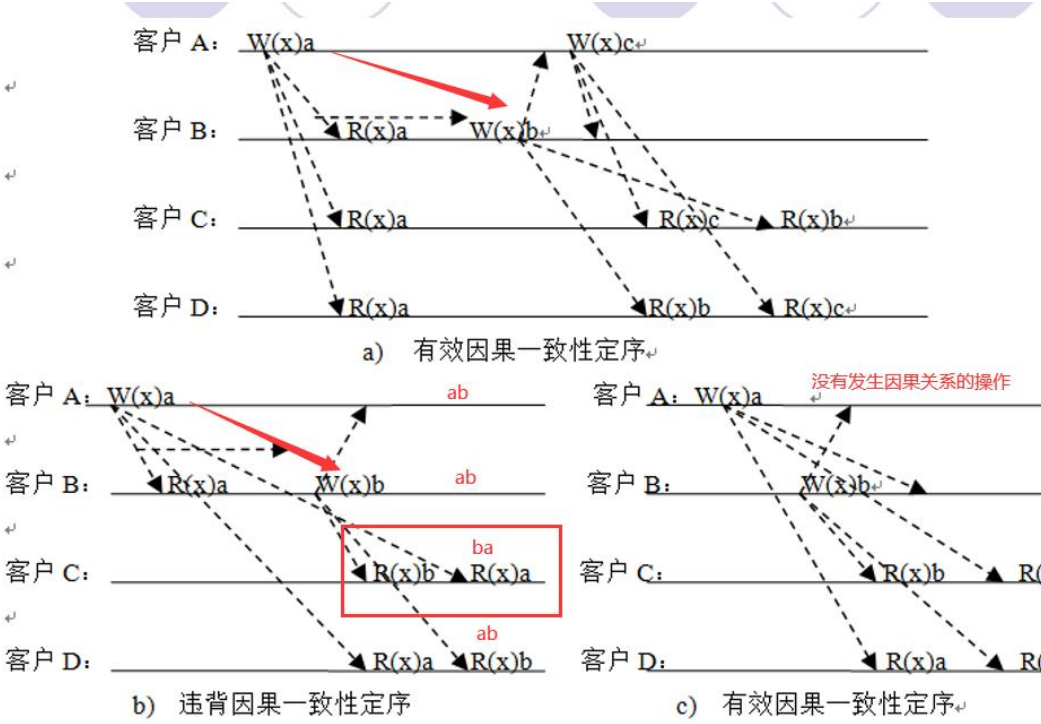
原因：C 看到的顺序是 ab，D 看到的顺序是 ba



8.4 图 8.3（c）为什么符合因果一致性定序

因果一致性：因果关系的**写操作**在所有的副本上上看到按同样的次序被执行

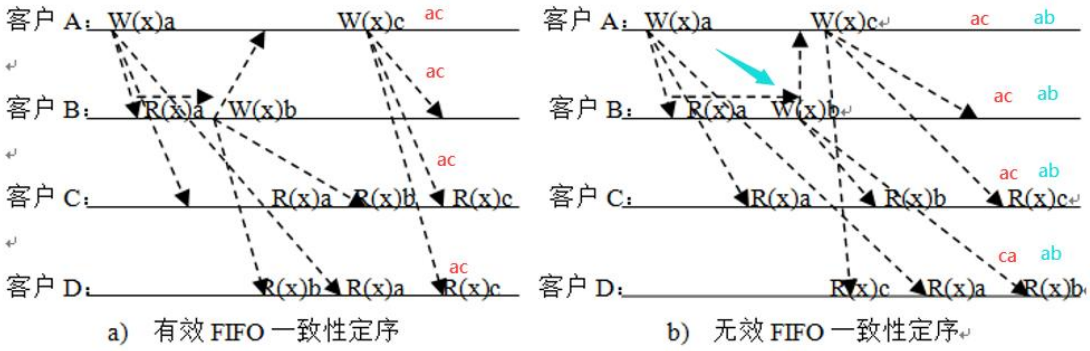
原因：在（c）中没有因果关系，是并发关系



8.5 图 8.4（b）为什么遵循因果一致性，但对 FIFO 一致性是无效的?

FIFO 一致性：只要求一个客户的写操作定序在所有副本上是相同的定序。

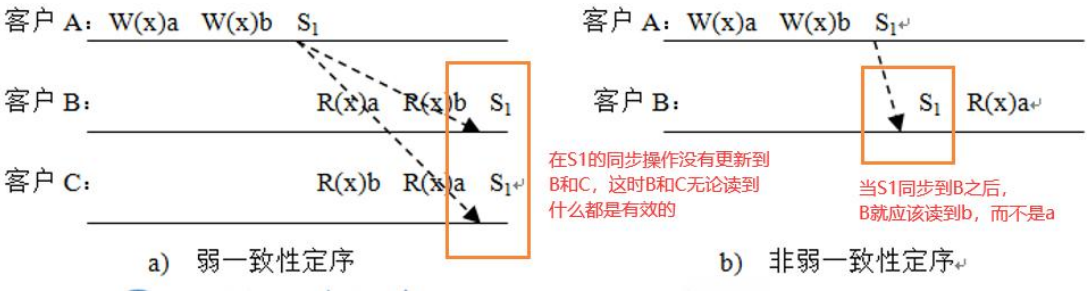
原因：（b）在每个副本上的因果操作定序都是 ab，而 FIFO 操作在客户 D 上的定序为 ca，和其他副本的定序不一致



a) 有效 FIFO 一致性定序　　b) 无效 FIFO 一致性定序

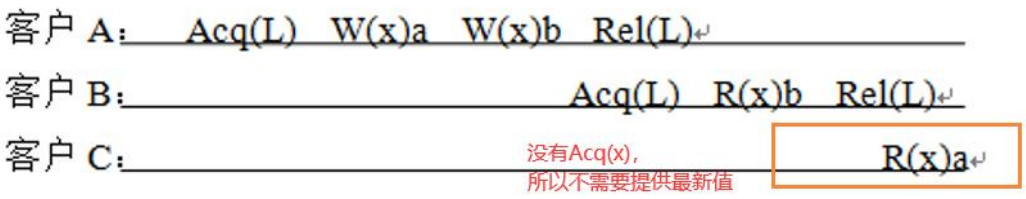
补充：

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。

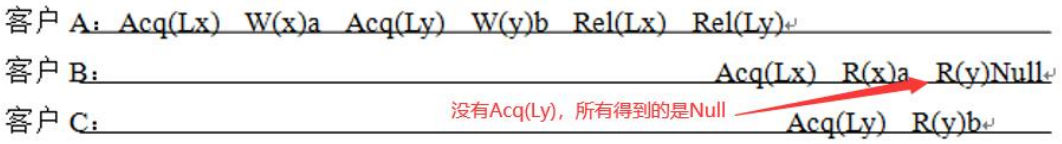


a) 弱一致性定序　　b) 非弱一致性定序

释放一致性：通过获取操作和释放操作，将**一组操作**与同步变量关联

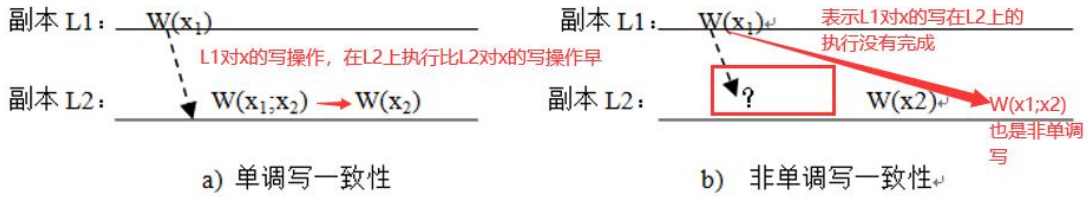


入口一致性：通过获取操作和释放操作，将**一次操作**与同步变量关联



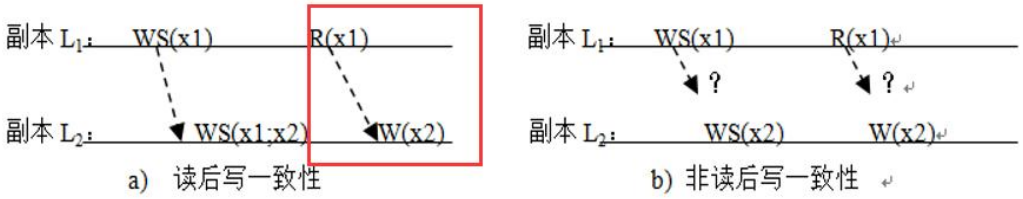
单调读：如果一个进程对数据项 x 的值，**该进程**的任何后续对 x 的对操作总是返回前一次读同样的值或更新的值。

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**。



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。

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，保证相同的或更新的 x 的值能被看见。



8.7 比较“传播更新通知”、“传播更新数据”和“传播更新操作”和他们的应用场合

传播更新通知：只传播一个简短的通知，不包含其他信息。

应用场合：当写操作对读操作的比率很高时，传播效果好。

传播更新数据：在副本间传播修改的数据。

应用场合：当读操作对写操作的比率很高时，传播效果好。

传播更新操作：在副本间传播执行的操作。要求每个副本有一个进程来执行更新操作，完成主动复制。

应用场合：当更新操作所关联的参数相对较少时，代价较少。

8.9 一个文件被复制在 10 个服务器上，列出表决算法的“读集团”和“写集团”

Gifford 表决算法：一个客户要读取一个具有N个副本的文件，必须将N_R个服务器组成一个“读集团”；要修改一个文件，必须将N_W个服务器组成一个“写集团”。N_R和N_W必须满足：

1 $N_R + N_W > N$　防止读写冲突（简单理解，画出的读写集团范围有交集）

2 $N_W > N/2$　防止写写冲突

根据要求$N_W > 5$，所以取$N_W = 6,7,8,9,10$

若$N_w = 6$，则N_R可以取 5,6,7,8,9,10

若$N_w = 7$，则N_R可以取 4,5,6,7,8,9,10

若$N_w = 8$，则N_R可以取 3,4,5,6,7,8,9,10

若$N_w = 9$，则N_R可以取 2,3,4,5,6,7,8,9,10

若$N_w = 10$，则N_R可以取 1,2,3,4,5,6,7,8,9,10

9.1 分布式文件系统分为远程访问模式和上载/下载模式，他们各适于何种环境？

远程访问模式：客户端可用存储不足，服务器提供了大量可用的操作接口

上载/下载模式：客户端可用存储充足，服务器仅提供读写操作

9.2 分布式文件系统的共享语义指的是什么，有几种共享语义？

共享语义：指的是多个用户同时访问共享文件的效果。其说明了被用户修改的数据何时可以被其他进程用户看到。

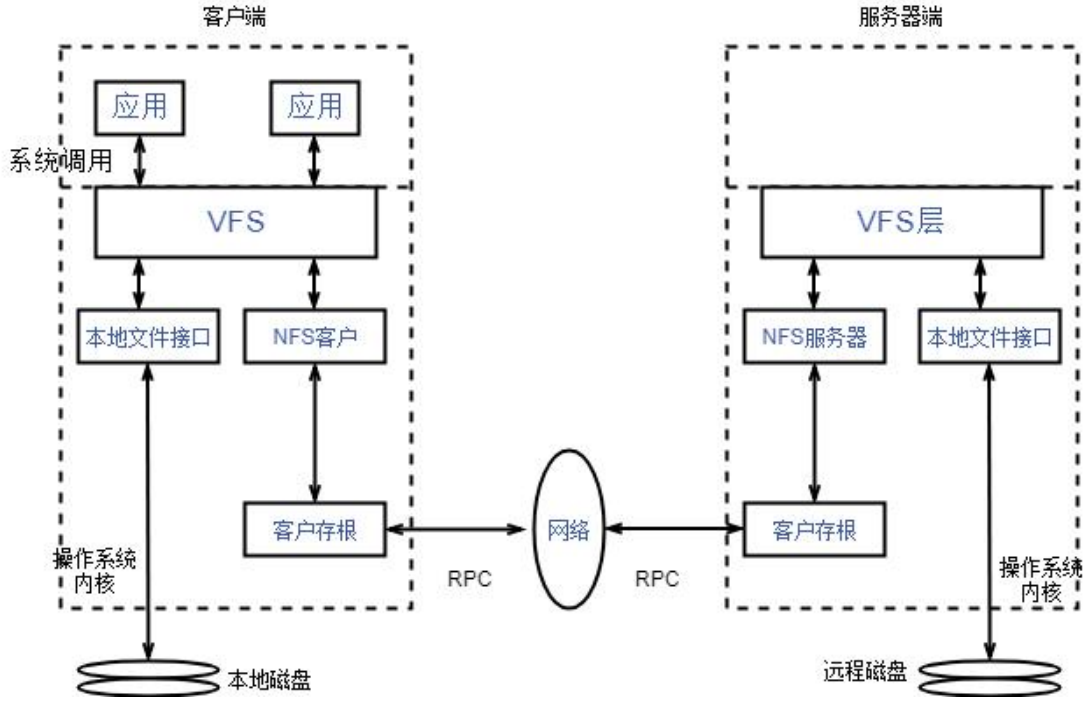
有四种共享语义：UNIX 语义、会话语义、不修改文件语义、事务语义

9.3 实现 NFSv3 的服务器必须是无状态的吗？

是的

9.4 NFS 是如何访问远程文件的，它的虚拟文件系统的作用是什么？

NFS 通过 RPC 访问远程文件。VFS 的作用是隐藏不同文件系统之间的差异，VFS 接口上的操作或传送到本地文件系统，或传送到 NFS 客户组件，由客户组件负责处理远程服务上的访问。



9.5 NFS 不提供全局共享名字空间，是否由模拟这种名字空间的方法？

为每个客户规定部分标准化的名字空间

9.6 根据第 8 章高速缓存协议，NFS 实现的是哪种高速缓存协议？

直写式高速缓存（不确定？）

9.7 NFS 是实现入口一致性还是实现释放一致性？

入口一致性（不确定）

9.8 NFS 用的是远程访问模式还是上载下载模式？

远程访问模式

9.9 NFS 的 RPC 实现了何种可靠性语义？

至少一次

11.6 什么是接口定义语言？它在分布式计算环境中起到什么作用？

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。

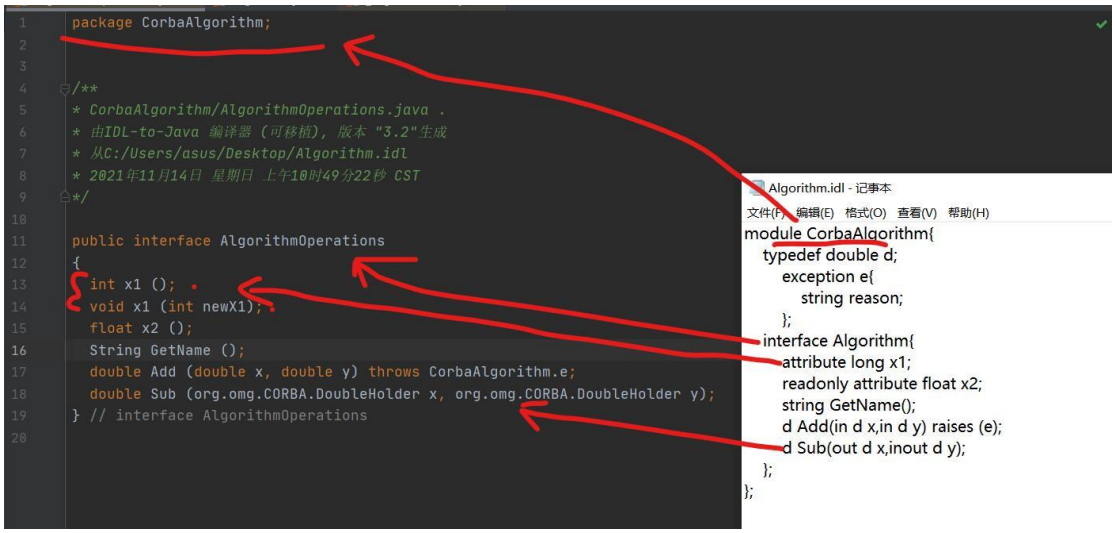
作用：实现标准的对象接口，构造分布式对象应用，使客户程序能调用远程服务器上对象的方法；IDL 为分布式对象系统定义模块、接口、类型、属性和方法提供了设施。

11.14 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);
    }

}
```

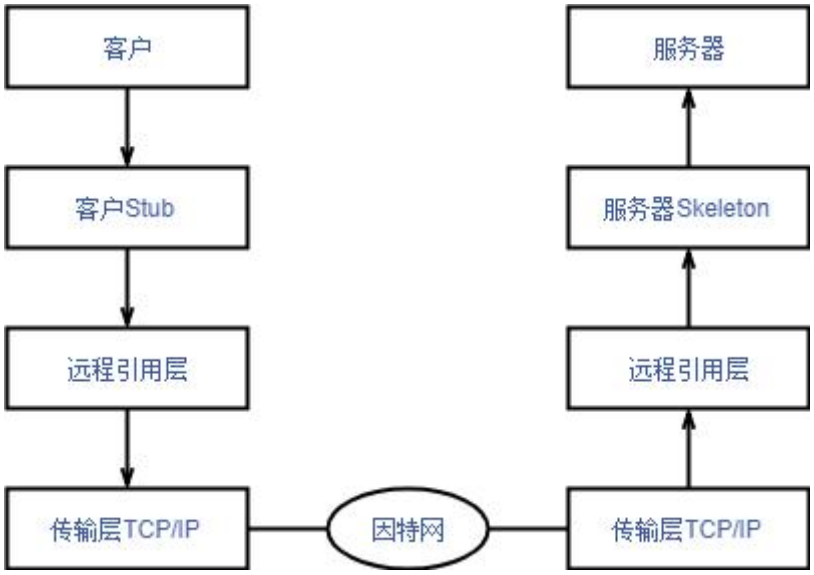
IDL 转 JAVA：

```
package BankingSystem;

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

package BankingSystem;

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



11.17 描述 JAVA RMI 的体系结构

RMI 是 JAVA 特有的分布式计算技术，由两个独立的程序组成：服务器和客户端。RMI 的体系结构如图所示，包括 3 层抽象：桩/骨架层、远程引用层、传输层。



11.18 简述 RMI 和 RPC 的关系

RMI 可以看做是 RPC 的 JAVA 实现，RMI 只支持 JAVA，是面向对象的分布式计算，支持对象的传递。

11.19 实现一个简单的 RMI 实例

远程接口 IRemoteMath.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
* 必须继承Remote类
* 所有参数和返回类型必须序列化（implements Serializable）（因为要网络传输）
* 任意远程对象必须实现此接口
* 只有远程接口中指定的方法可以被调用
*/
public interface IRemoteMath extends Remote{

    // 所有方法必须抛出RemoteException
    public double add(double a, double b) throws RemoteException;
}
```

远程接口实现类 RemoteMath.java

```
import java.rmi.RemoteException;
import java.rmi.UnicastRemoteObject;

/**
* 服务器端实现远程接口
* 必须继承UnicastRemoteObject，以允许JVM创建远程的代理
*/
public class RemoteMath extends UnicastRemoteObject implements IRemoteMath{

    @Override
    public double add(double a, double b) throws RemoteException{
        return a+b;
    }
}
```

服务器端 RMIServer.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

/**
* 创建RemoteMath类的实例，并在rmiregistry中注册
*/
public class RMIServer{
    public static void main(String[] args){
        try{
            // 注册远程对象，向客户端提供远程对象服务
            // 远程对象是在远程服务上创建的，你无法确切地知道远程服务器上的对象的名称
            // 但是，将远程对象注册到rmi registry之后
            // 客户端就可以通过rmi registry请求到该远程对象的stub
            // 利用stub代理就可以访问远程对象了
            IRemoteMath remoteMath = new RemoteMath();
            LocateRegistry.createRegistry(1099);
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Compute", remoteMath);
            // 如果不想再让对象被继续调用，使用下面这行
            // UnicastRemoteObject.unexportObject(remoteMath, false);
        }catch(Exception e){}
    }
}
```

客户端 MathClient.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.Naming;

public class MathClient{
    public static void main(String[] args){
        try{
            // 如果RMI Registry就在本地，URL就是：rmi://localhost:1099/Compute
            // 否则，URL就是：rmi://RMIService_IP:1099/Compute
            Registry registry = LocateRegistry.getRegistry("localhost");
            // 从Registry中检索远程对象的代理
            IRemoteMath remoteMath = (IRemoteMath)registry.lookup("Compute");
            // 获取代理的另一种方法
            // IRemoteMath remoteMath = (IRemoteMath)Naming.lookup("rmi://localhost:1099/Compute");
            // 调用远程对象的方法
            double res = remoteMath.add(5.0, 3.0);
        }catch(Exception e){}
    }
}
```

12.6 什么是 Web 契约，它要说明什么问题？

Web 契约：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中，契约是系统之间交换数据时应遵循的约定。

说明的问题：

服务功能描述（What）：服务的目的和能力是什么

服务访问描述（How）：如何访问服务

服务位置描述（Where）：服务在何处被访问

12.7 什么是服务的抽象描述？什么是服务的具体描述？他们描述的内容是什么？

抽象描述：服务功能描述

具体描述：服务访问描述+服务位置描述

服务功能描述：（POMT）端口类型定义、操作定义、消息定义、类型定义、策略定义

服务访问描述：（POM）端口类型绑定、操作绑定、消息绑定、策略定义

服务位置描述：（SPA）服务定义、端口定义、地址定义、策略定义

12.8 什么是 WDSL？它与 XML 有什么关系？WSDL 用来描述什么实体？

WSDL：即 web 服务描述语言。

关系：

1.wsdl 是一个基于 XML 的语言，它描述了和特定 Web 服务之间的交互机制，并且使用该语言可以约束服务提供者以及使用服务的所有请求者

2.wsdl 是一个基于 XML 的规范模式， 提供了一个标准的服务表示语言，可用于描述 Web 服务所暴露的公共接口细节

WSDL 用于描述网络服务。

12.9 什么是 SOAP？它与 HTTP 和 WSDL 有什么关系？

SOAP：简易对象访问协议。

SOAP 是由 HTTP 承载，SOAP 请求文档是 HTTP 的请求体。SOAP 方法是一个 HTTP 请求/响应。（GET/POST 都可以）

关系：SOAP 是个通信协议，SOAP 在 HTTP 协议的基础上，把编写成 XML 的 REQUEST 参数，放在 HTTP BODY 上提交给 WEB SERVICE 服务器。WEB SERVICE 服务器处理完成后，把结果也写成 XML 作为 RESPONSE 送回用户端，为了使用户端和 WEB SERVICE 可以相互对应，可以使用 WSDL 作为这种通信方式的描述文件，利用 WSDL 工具可以自动生成 WS 和用户端的框架文件。

12.10 什么是 UDDI？它与 HTTP、WSDL 和 SOAP 的关系是什么？

UDDI：通用描述、发现和集成

关系：

WSDL 用来描述服务；

UDDI 用来注册和查找服务；

SOAP 作为传输层，用来在消费者和服务提供者之间传送消息。

用户可以在 UDDI 注册表（registry）查找服务，取得服务的 WSDL 描述，然后通过 SOAP 来调用服务，同时 UDDI 采用的也是 HTTP 协议。

补充：SOAP 报文

一个 GetStockPrice 请求被发送到了服务器。此请求有一个 StockName 参数，而在响应中则会返回一个 Price 参数。此功能的命名空间被定义在此地址中："http://www.example.org/stock"

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

12.13 什么是 SOA 概念框架？SOA 试图解决什么问题？

SOA 面向服务的体系结构，是一个组件模型，它将应用程序的不同功能单元称为服务，这些服务之间通过定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。

SOA 的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。

试图解决的问题：

（1）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

（2）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

（3）企业 IT 系统抽象程度低，与业务之间存在着断层。

（4）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

12.16 BPEL 定义了哪些活动？（P404）

基本活动

结构化活动

12.17 试述 BPEL、SCA 和 SDO 之间的关系

在 SOA 架构下，BPEL 是业务流程执行语言，关注流程，把各个服务按需串联起来（12.16）；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（12.14）；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（12.15），他们三者组成 SOA 的铁三角。