分布计算环境

北京邮电大学计算机学院

Chapter 2

分布式系统的基本原理

主要内容

- 体系结构
- 进程
- 通信
- 命名
- 一致性和复制
- 容错
- 安全

引言

- 分布式系统往往是由各种复杂的系统组成，其组件按照定义分散在多台机器中。要掌握这些复杂性，关键是恰当地组织好这些系统，分析其结构
  - **软件体系结构**告诉我们不同的软件组件在逻辑上是如何组织的，它们之间是如何相互作用的
  - 当我们把系统的各个软件组件部署到各个物理机器上时，同一系统可以有不同的部署方式，而软件体系结构部署后的最终实例称为**系统体系结构**

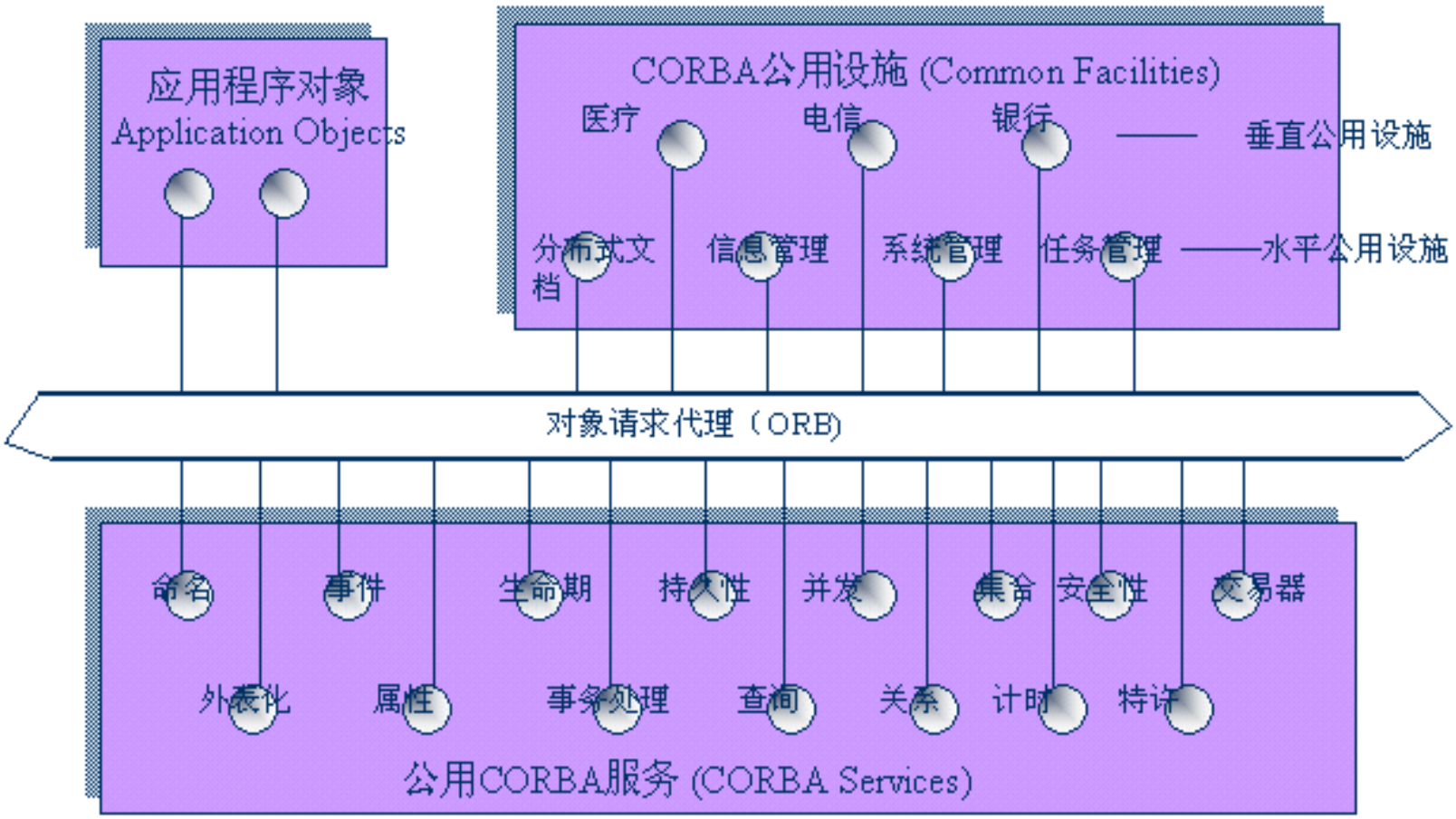
体系结构主要内容

- 软件体系结构
- 体系结构样式
- 系统体系结构
- 体系结构与中间件

软件体系结构

- 某定义：软件体系结构（软件架构）是具有一定形式的结构化元素，即：组件的集合，包括：处理组件、数据组件和连接组件。处理组件负责对数据进行加工，数据组件是被加工的信息，连接组件把体系结构的不同部分组合连接起来
  - 组件：是一个模块单元，它具有可以提供良好定义的接口，在其环境中是可替换的
  - 连接组件（连接件）：描述为一种机制，在组件之间传递通信、使组件相互协调和协作

软件体系结构例：CORBA



体系结构主要内容

- 什么是体系结构
- 体系结构样式
- 系统体系结构
- 体系结构与中间件

什么是软件体系结构样式

- 某定义：软件体系结构样式（Style，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。它反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统
- 按这种方式理解，软件体系结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

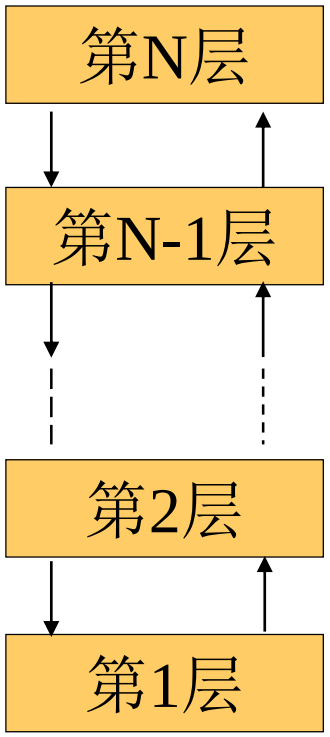
分布式系统中的重要样式

- 1、分层体系结构
- 2、面向对象的体系结构
- 3、以数据为中心的体系结构
- 4、基于事件的体系结构

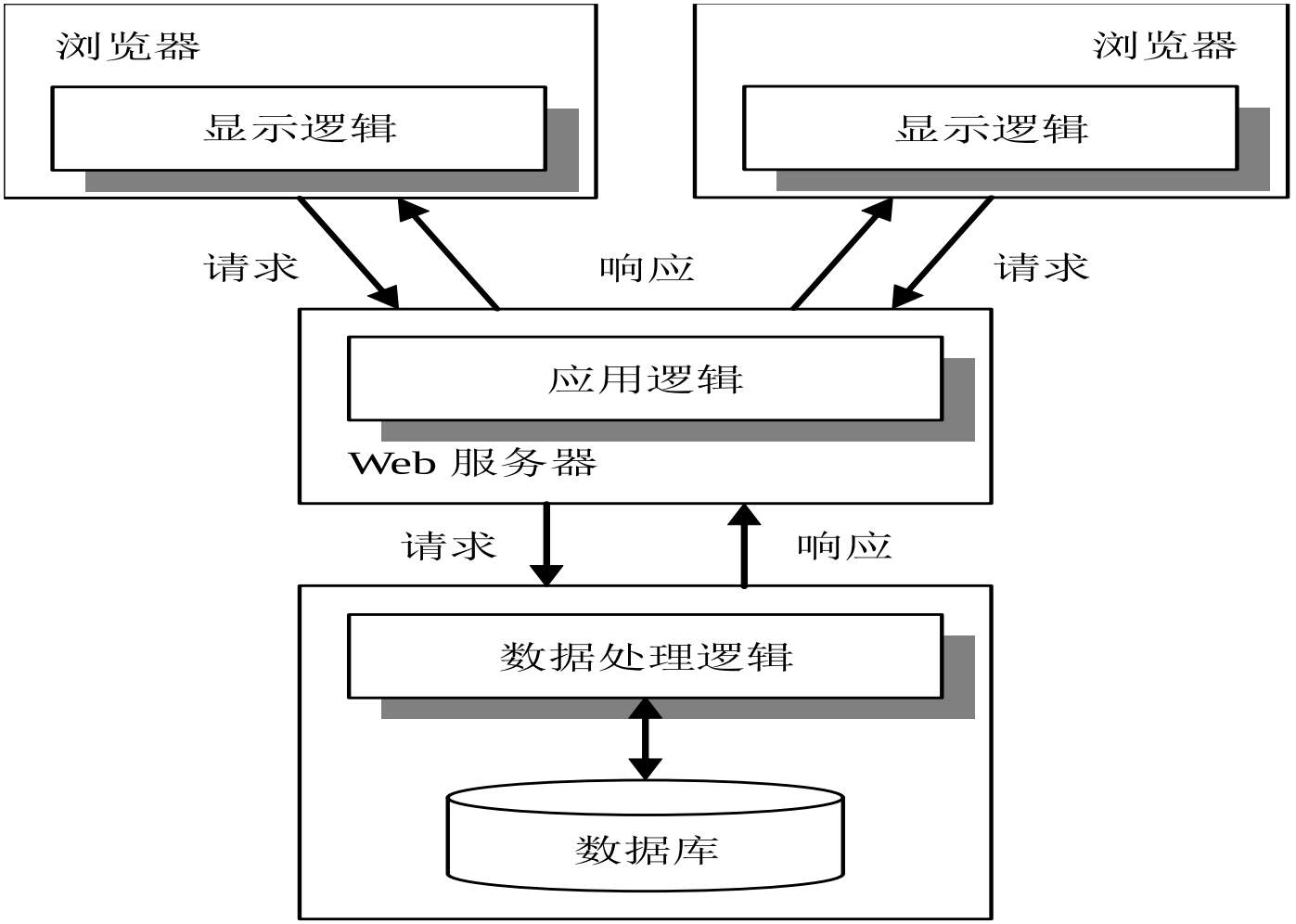
分层体系结构

- 层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见

1. 系统中组件在一些层实现了虚拟机
2. 连接件通过决定层间如何交互的协议来定义
3. 拓扑约束包括对相邻层间交互的约束

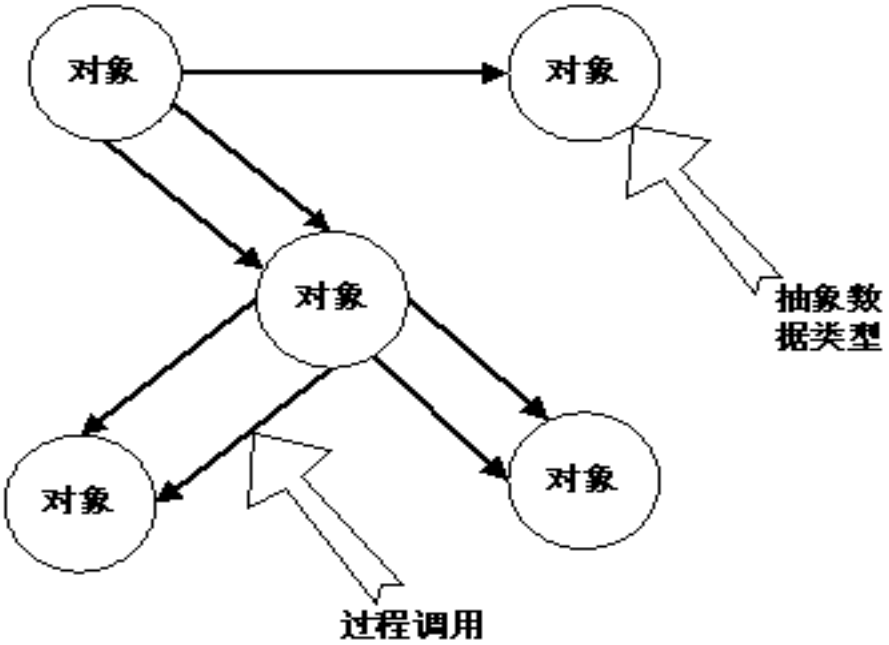


分层体系结构例：三层B/S结构

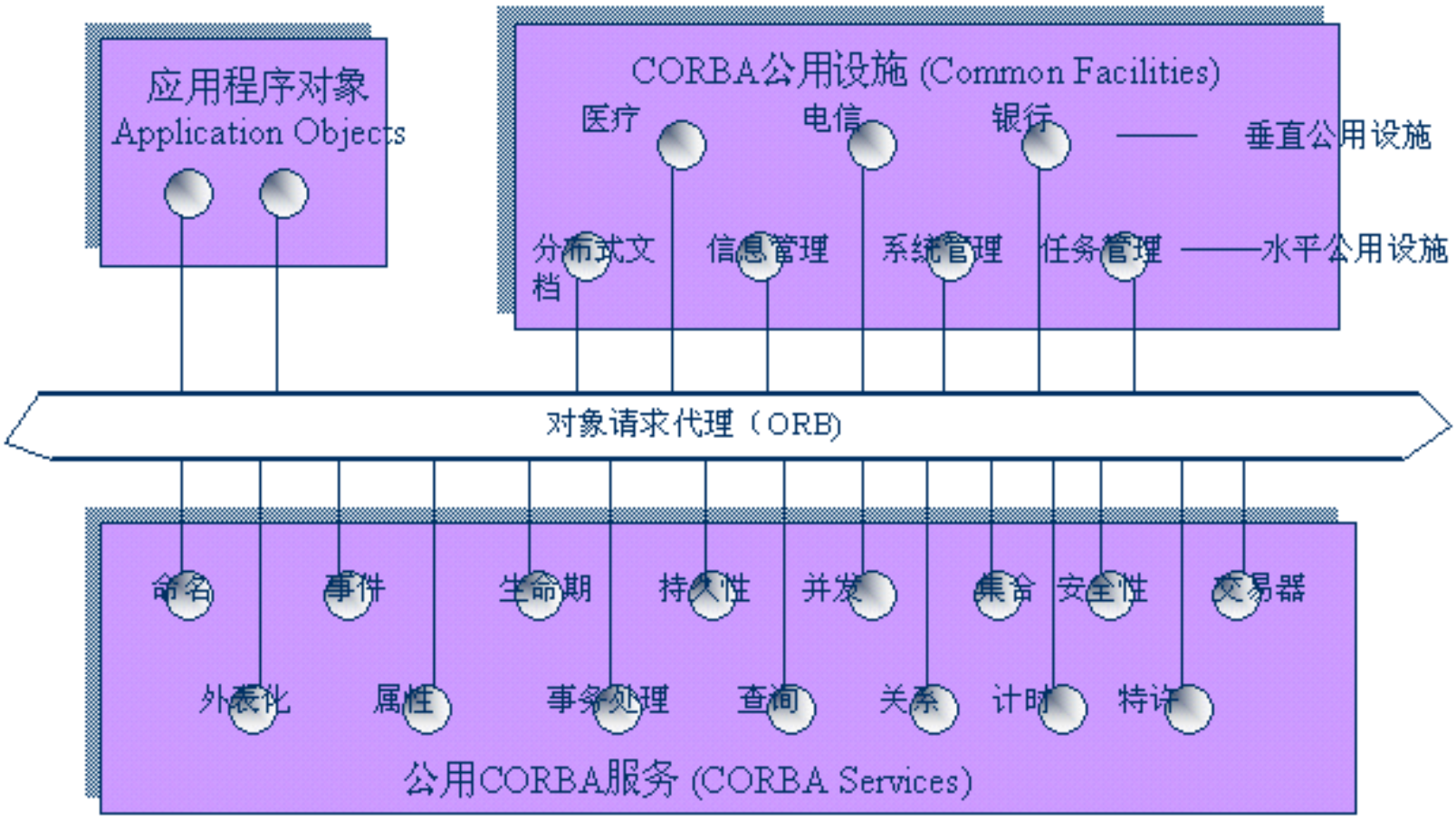


抽象数据类型和面向对象的体系结构

- **抽象数据类型ADT：是指一个数学模型以及定义在此数学模型上的一组操作**
  - 即：数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中
- 这种风格的组件是对象，或者说是抽象数据类型的实例
- 与“客户-服务器”结构很相似

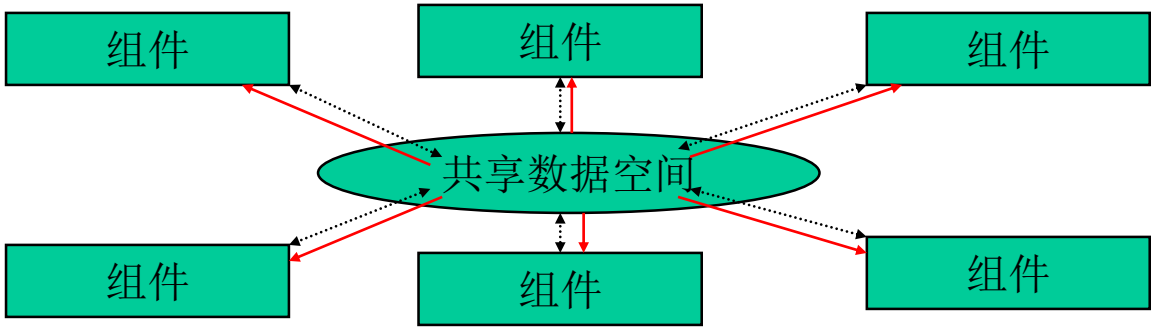


面向对象的软件体系结构例：CORBA



以数据为中心的体系结构

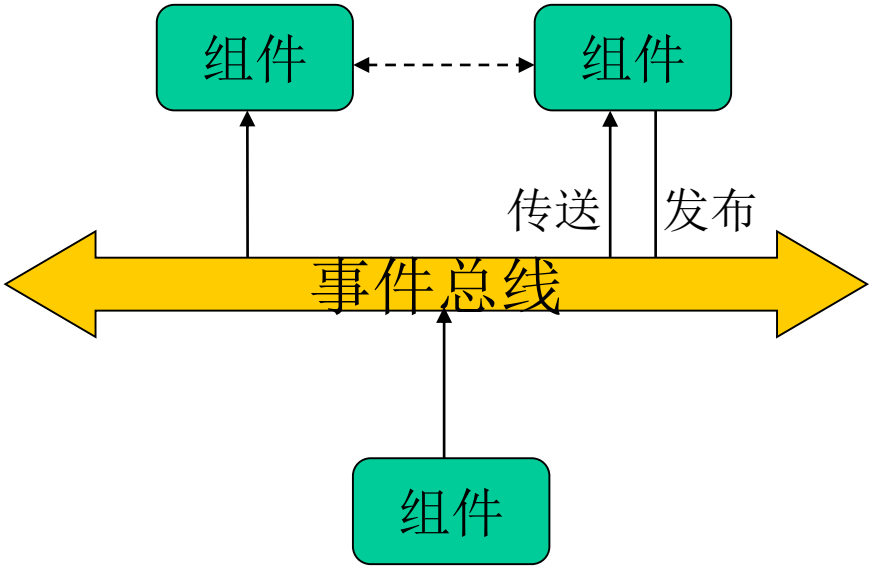
- 组件间的通信，基于一个公用的仓库（如共享的分布式文件系统）实现
  - 被动仓库式：仓库数据被动地被组件读写
    - 如：传统数据库系统、Web系统
  - 主动仓库式：仓库的当前状态触发并选择组件需要执行的过程
    - 如：拼图游戏

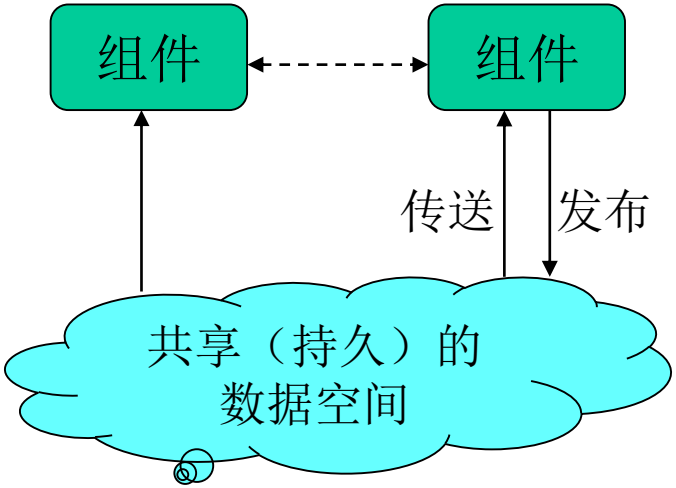


主动仓库式（黑板式）

基于事件的体系结构

- 组件间的通信，通过事件（可带有数据）的传播实现，如
  - 发布/订阅(publish/subscribe)系统:松耦合
  - 与以数据为中心的体系结构组合构成的共享数据空间样式

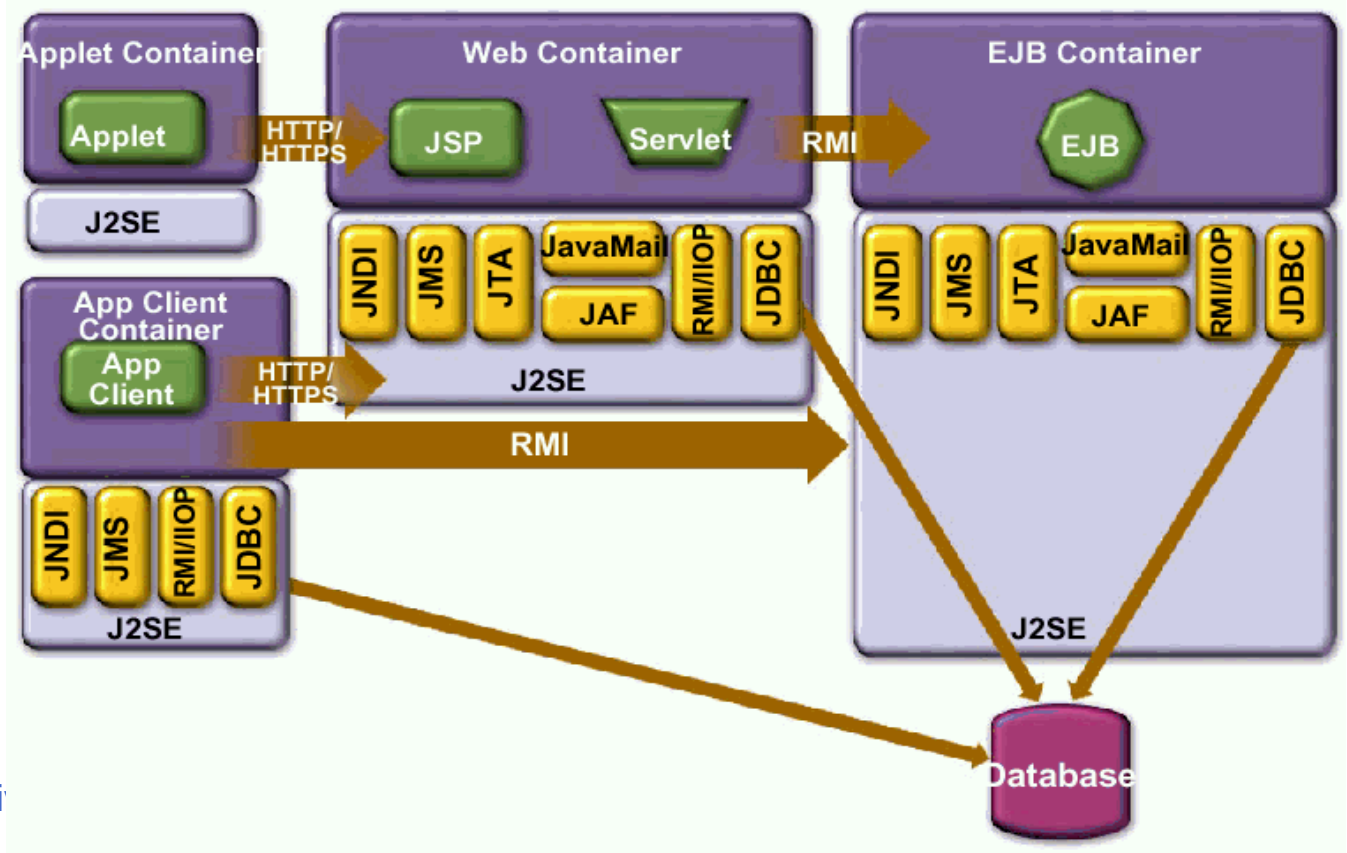




体系结构样式的混合

- 实际的分布式系统，常常混合了两种乃至多种样式

- 1、分层体系结构
- 2、面向对象的体系结构
- 3、以数据为中心的体系结构
- 4、基于事件的体系结构



体系结构主要内容

- 软件体系结构
- 体系结构样式
- 系统体系结构
- 体系结构与中间件

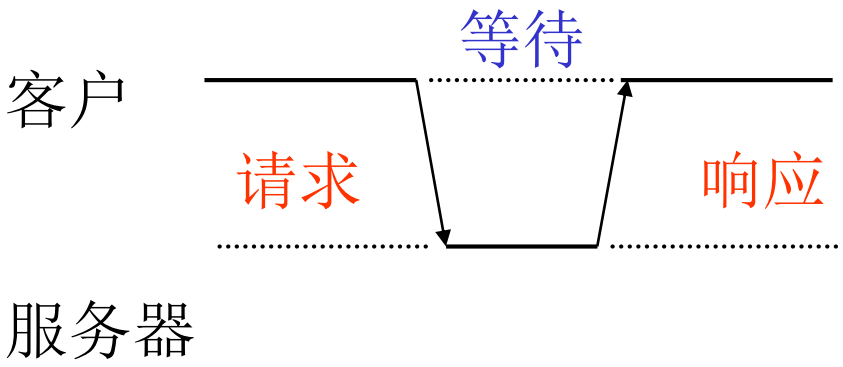
系统体系结构

- 系统体系结构：软件体系结构的具体实例。确定了软件组件、这些组件的交互以及它们的位置（部署）就是软件体系结构的一个实例

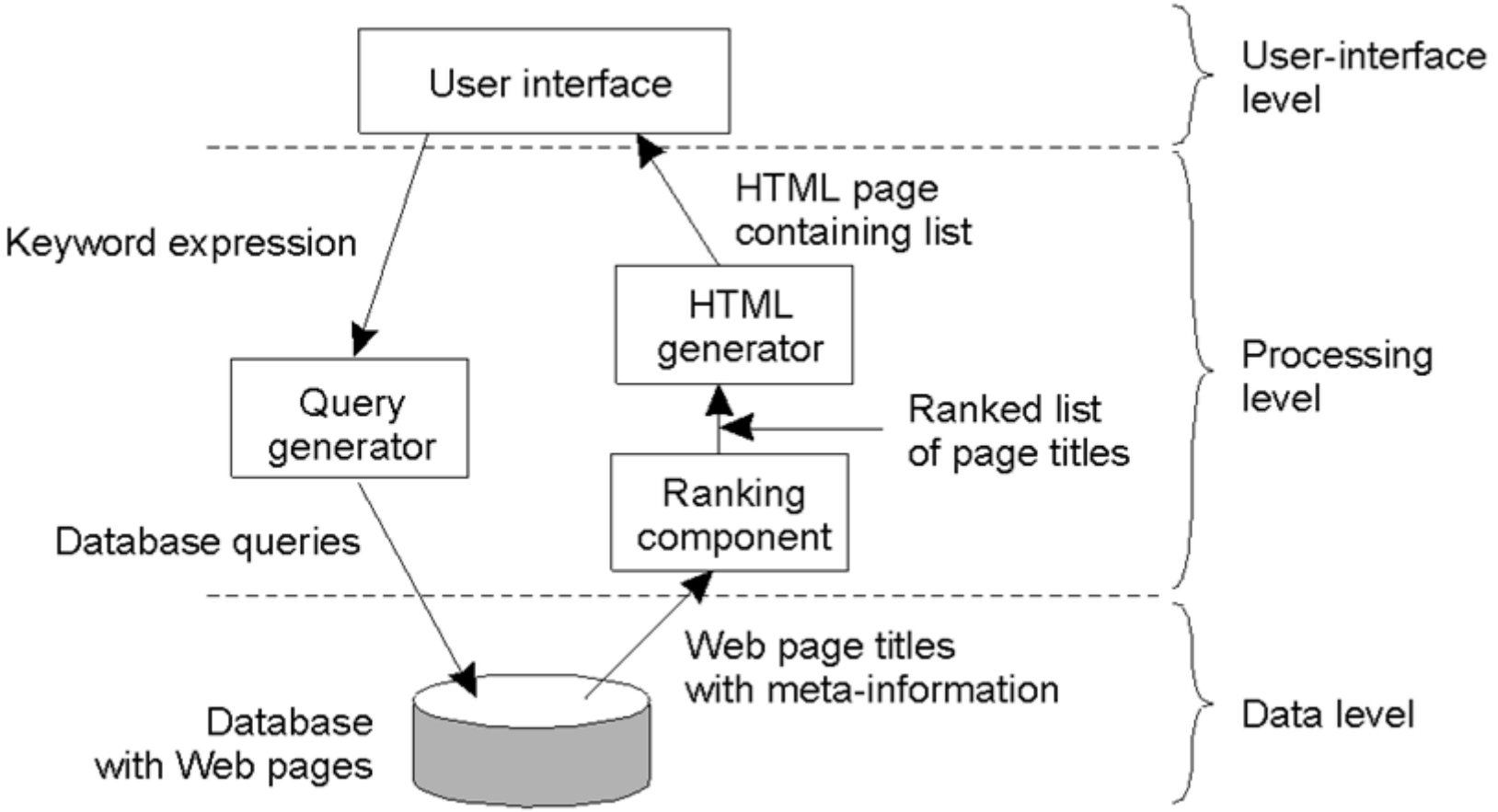
- 主要讨论
  1、集中式体系结构
  2、非集中式体系结构
  3、混合体系结构

集中式体系结构

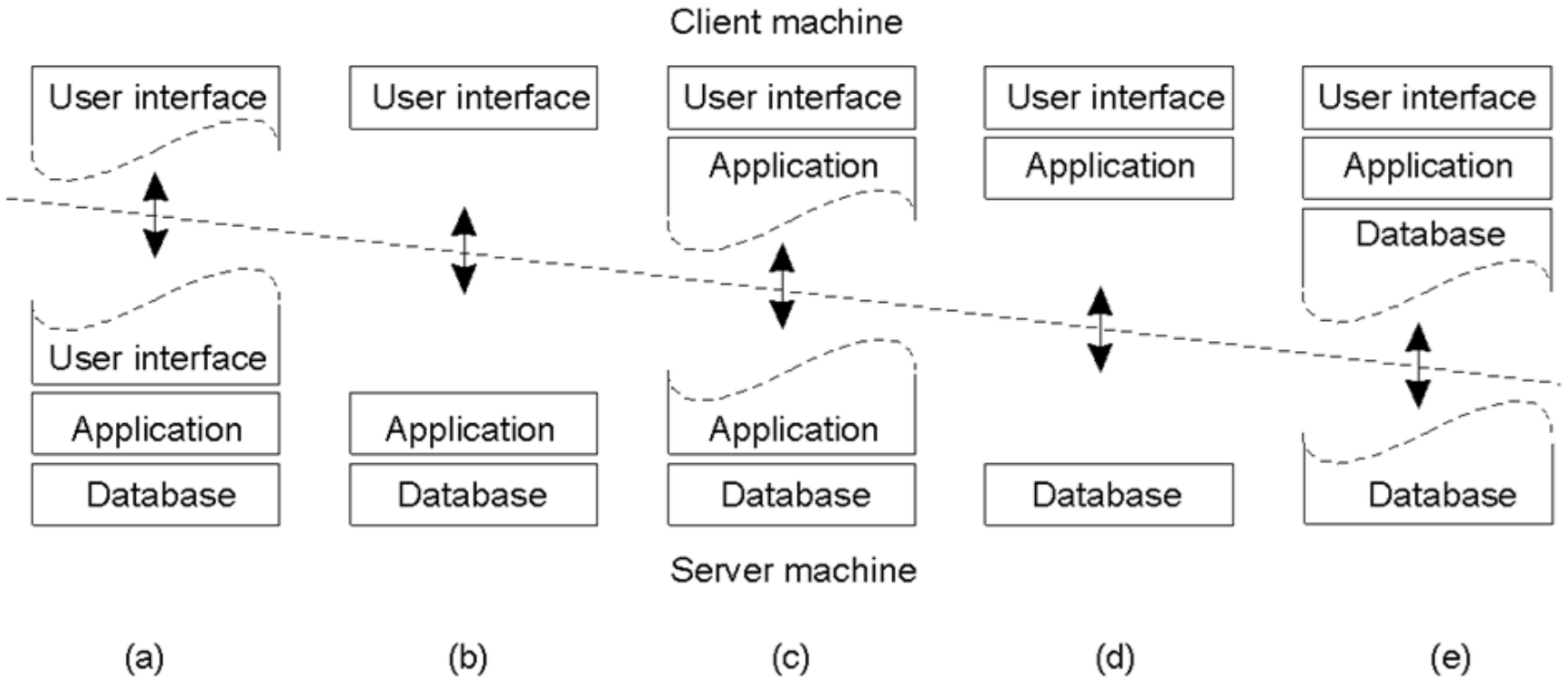
- 顾名思义，服务功能集中在服务器侧
- 客户/服务器体系结构
  - 服务器：实现特定服务的进程
  - 客户：向服务器提出请求、等待答复的进程
  - 请求/答复模式



集中式的层次例：搜索引擎



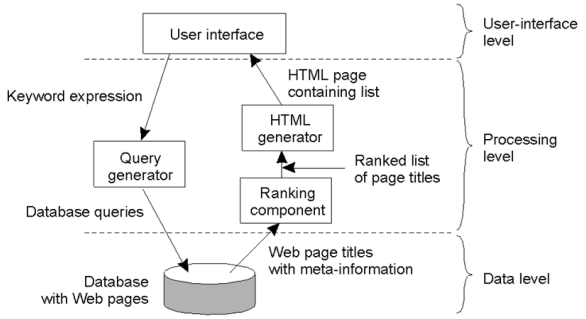
各种各样的分层方式



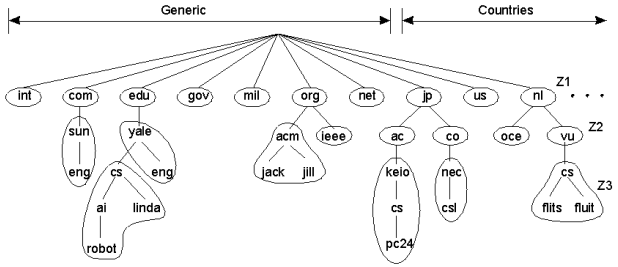
如何分层取决于应用系统的具体情况

非集中式体系结构

- 多层客户-服务器结构 --- 垂直分布性
  - 将不同功能组件放在不同的机器上
  - 有助于功能可以从逻辑、物理上分割在多台机器上，每台机器按特定功能定制

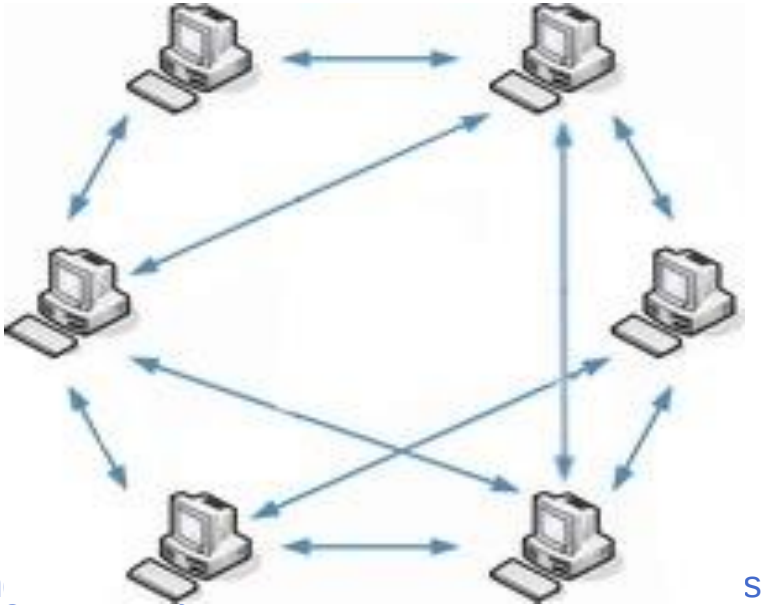


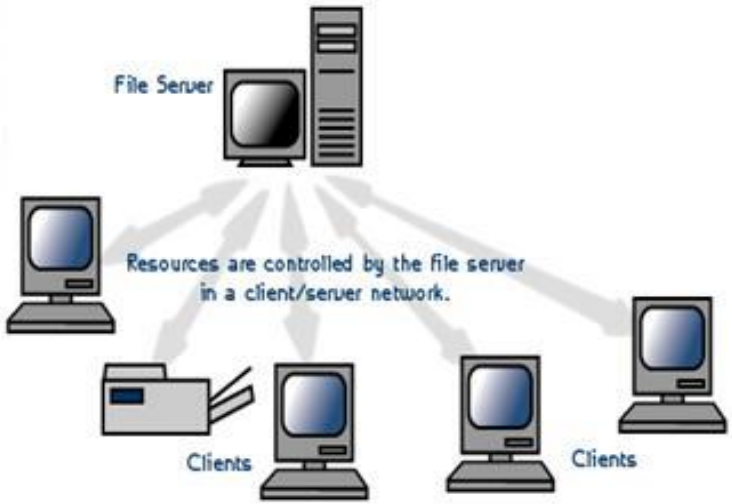
- 另一种体系结构 --- 水平分布性
  - 在物理上被分割成逻辑相等的几个部分
    - ➔每个部分都处理整个数据集中自己共享的部分
  - 进一步地：点对点（P2P）



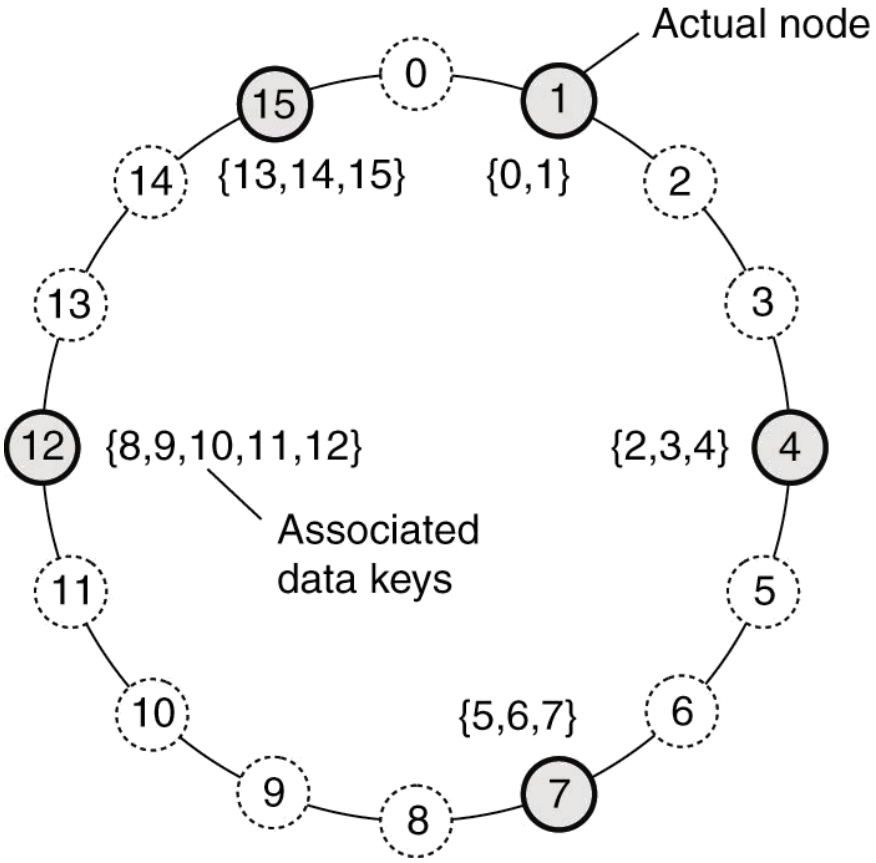
从集中式到P2P

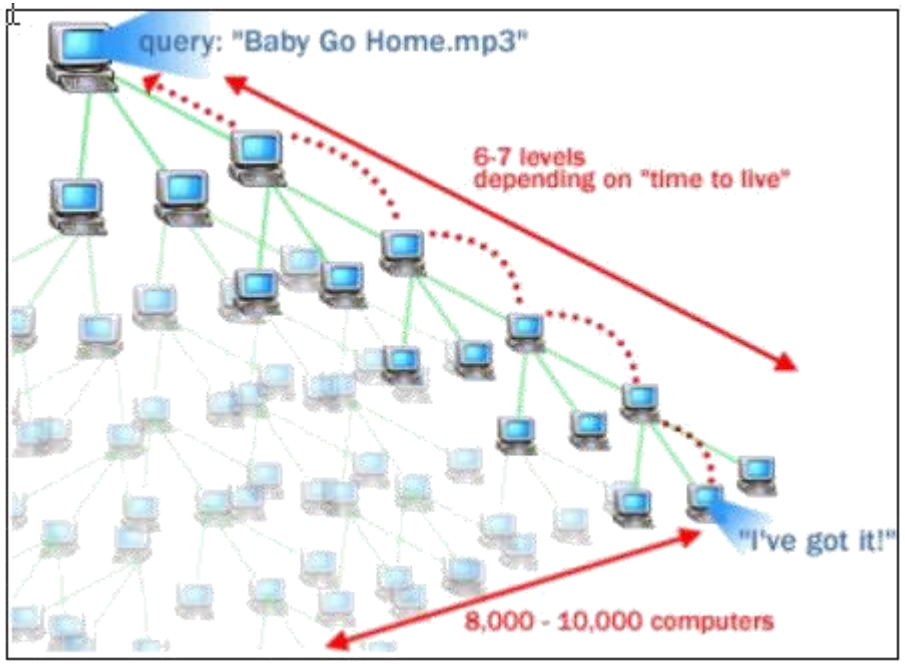
♦**P2P** 是一种分布式网络，网络的参与者共享他们所拥有的一部分硬件资源（处理能力、存储能力、网络连接能力等）和软件资源或者数据资源，这些共享资源能被其它对等节点（**Peer**）直接访问而无需经过中间实体。在此网络中的参与者既是资源提供者（**Server**），又是资源获取者（**Client**）





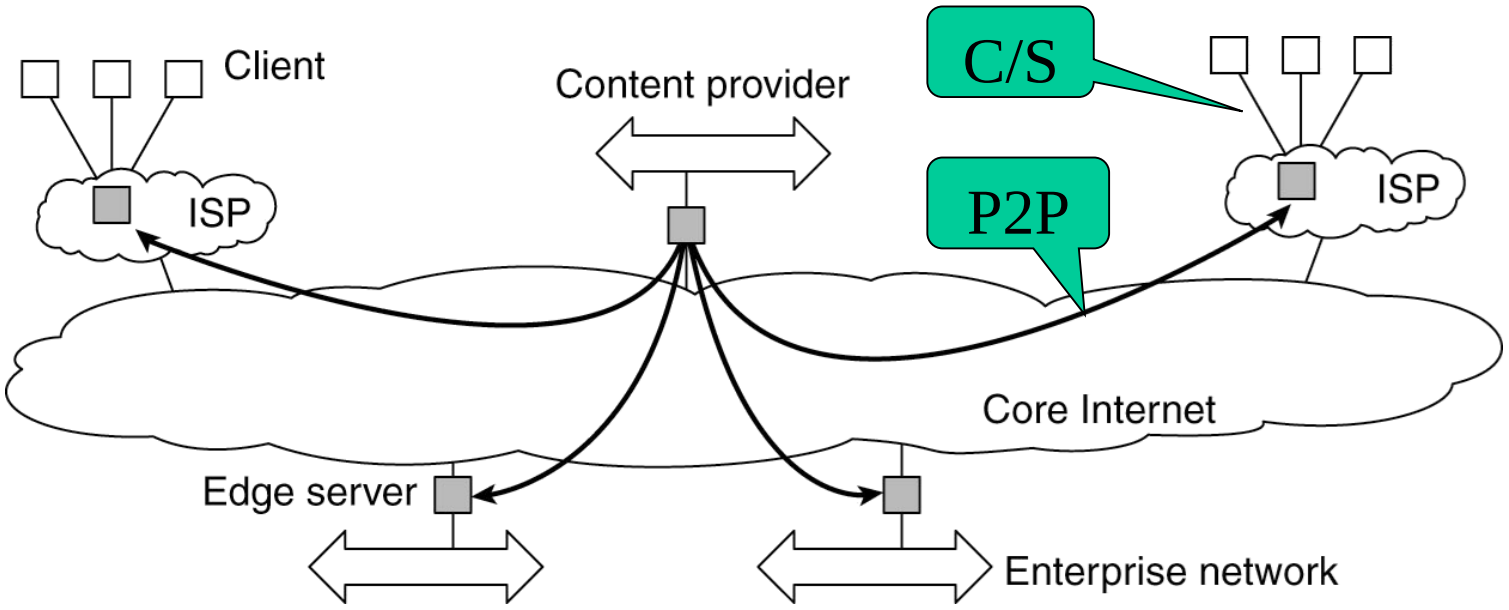
P2P网络的拓扑结构示例





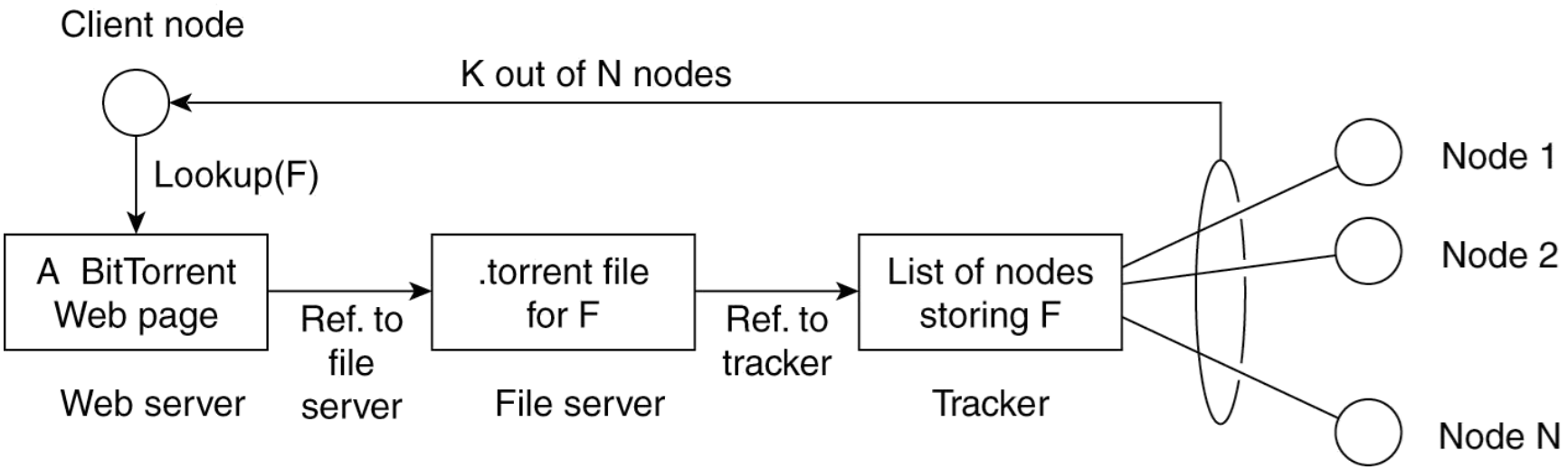
混合型体系结构

- 将客户/服务器结构与非集中式结构相结合
- 如：边缘服务器系统（edge server）



混合型体系结构（2）

- 再如：协作式分布式系统
- 例：文件共享系统 BitTorrent
  - 强制每个参与者，即可下载文件，也负责上载文件
  - 全局目录：在Web站点中保存，基于它，可找到相应的.torrent文件，并进一步找到相应的tracker。
  - 跟踪器（tracker）:记录活动节点的服务器

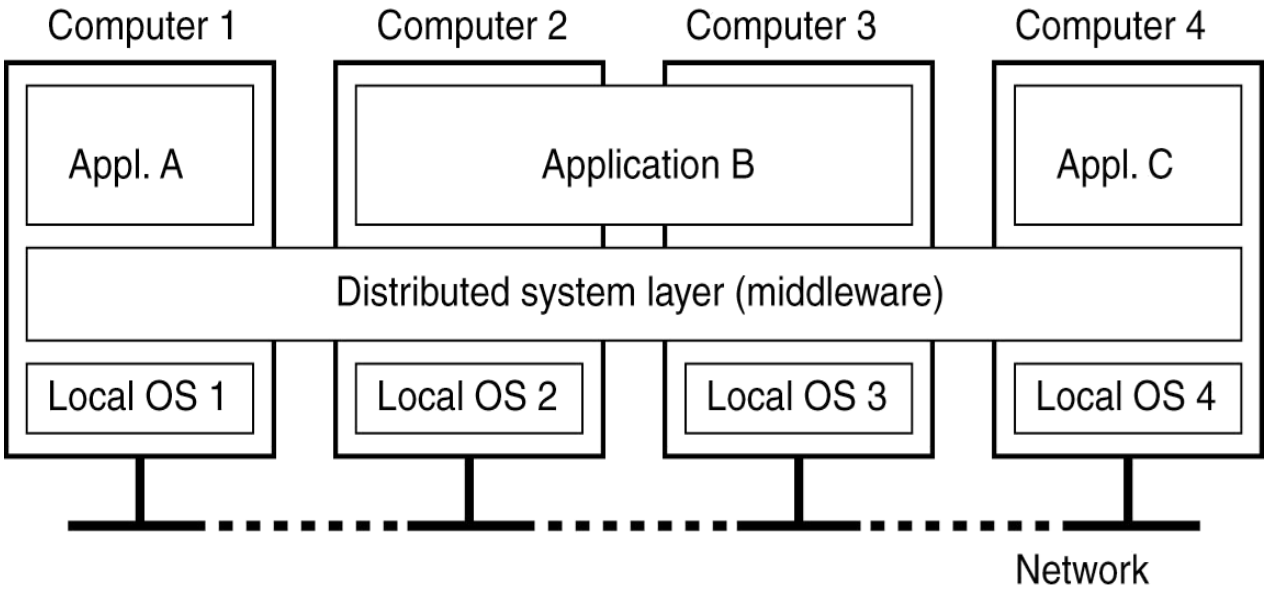


体系结构主要内容

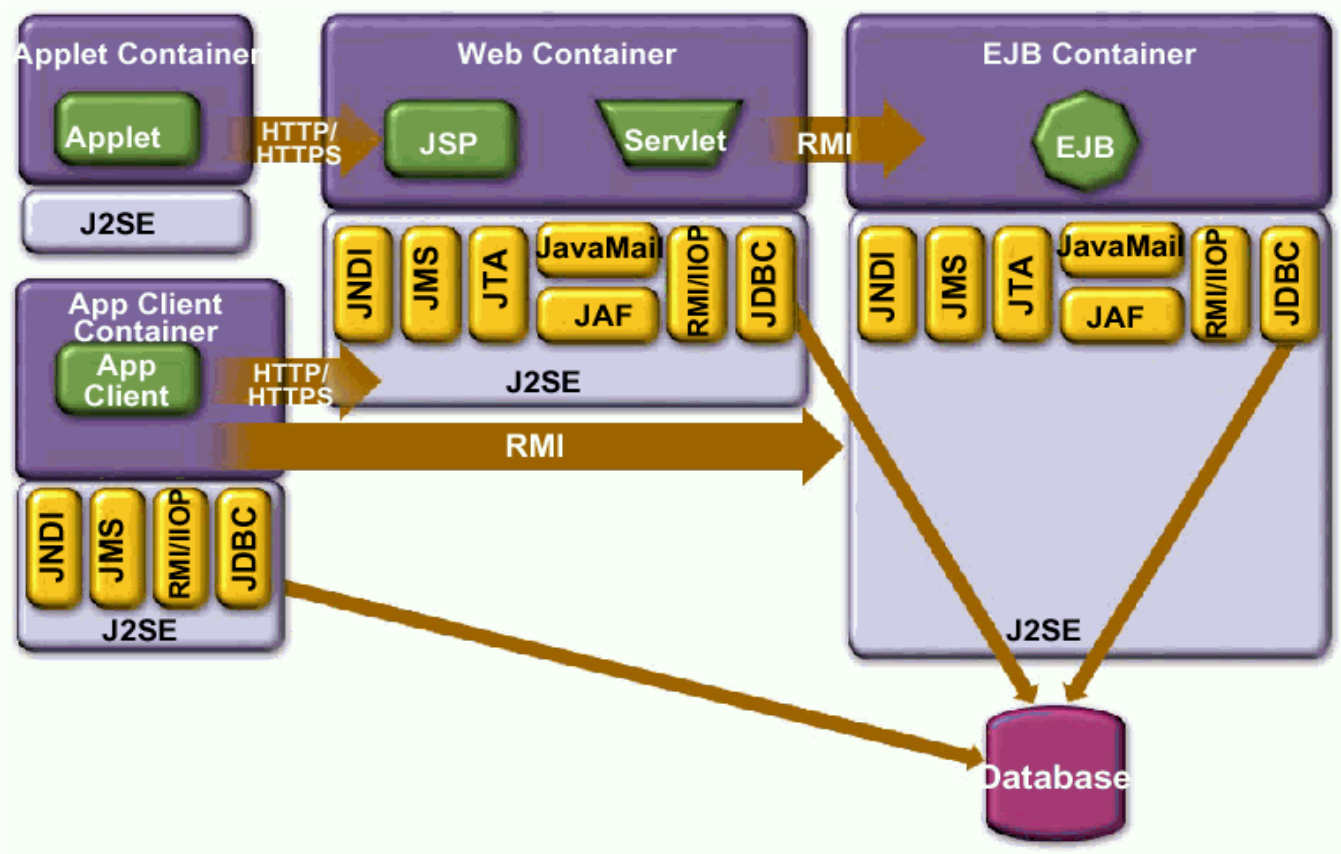
- 软件体系结构
- 体系结构样式
- 系统体系结构
- 体系结构与中间件

体系结构与中间件

- 中间件是操作系统和应用软件之间的一个独立软件层，它在不改变现有操作系统的前提下，向分布式应用提供相应的执行环境和编程环境
- 分布式计算环境就是（一种）中间件，能够屏蔽操作系统和网络协议的差异，能够为异构系统之间提供通讯服务以及各种通用服务的软件

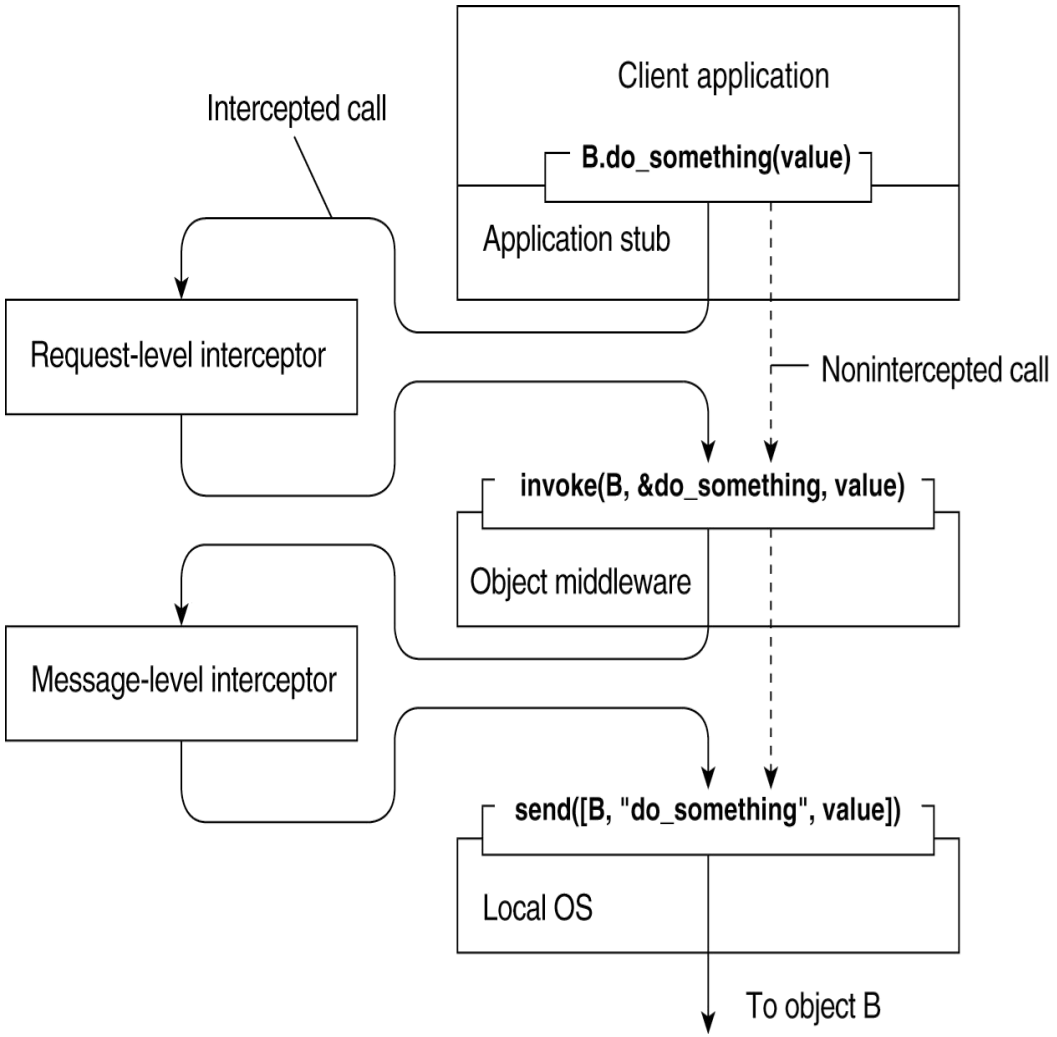


Java EE中的中间件



中间件中的拦截器

- 拦截器（Interceptor）：中间件中大多具有的一种组件，方便中间件的配置和定制
  - 可中断正常执行的控制流，插入执行其他代码
- 例：远程对象调用
  - 请求级拦截器
  - 消息级拦截器



主要内容

- 体系结构
- 进程
- 通信
- 命名
- 一致性和复制
- 容错
- 安全

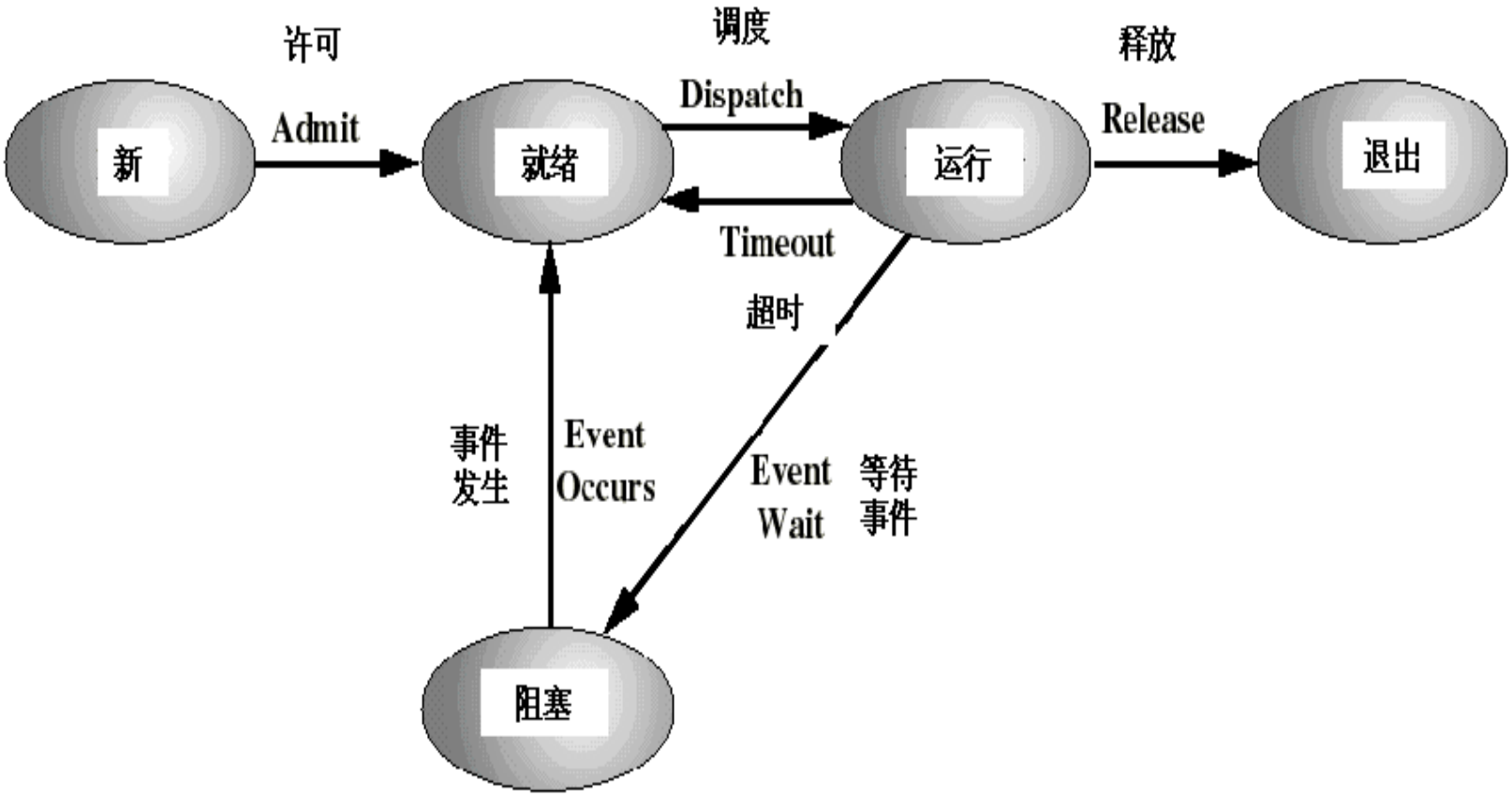
进程主要内容

- 进程
- 线程
- 客户
- 服务器

进程

- 进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动
  - 进程是系统资源分配的基本单元
  - 在早期面向进程设计的计算机结构中，是系统基本的执行单元或者说调度单元
  - 在当代面向线程设计的计算机结构中，是线程的容器
- 多个不同的进程可以包含相同的程序：一个程序在不同的数据集里就构成不同的进程，能得到不同的结果；但是执行过程中，程序不能发生改变

五状态进程模型


许可
Admit
调度
Dispatch
释放
Release
新
就绪
运行
退出
Timeout
超时
事件
发生
Event
Occurs
Event
Wait
等待
事件
阻塞

进程的特点

- 动态性：进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的
- 并发性：任何进程都可以同其他进程一起并发执行
- 独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位
- 异步性：由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进
- 结构特征：进程由程序、数据和进程控制块三部分组成。

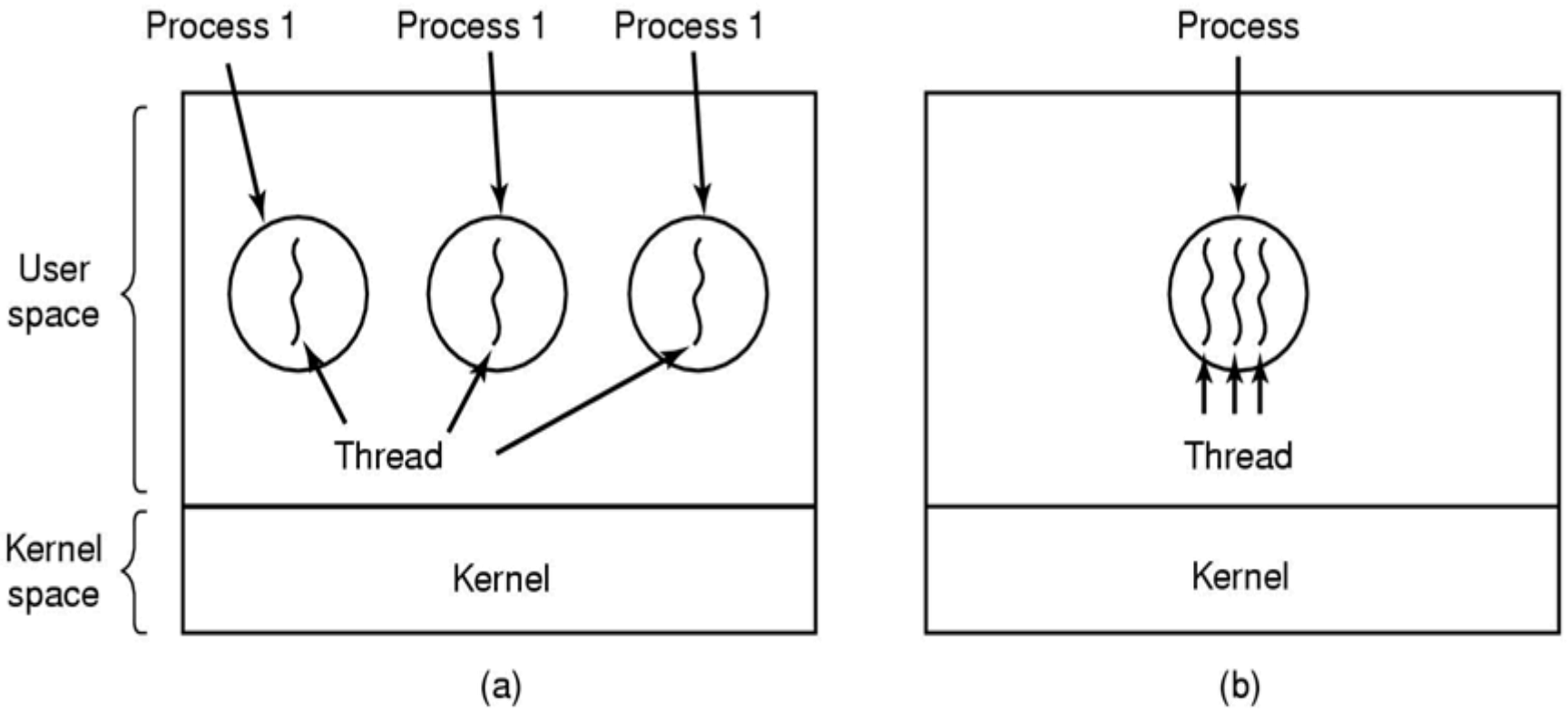
进程主要内容

- 进程
- 线程
- 客户
- 服务器

线程的引入

- 进程的引入提高了计算机资源的利用效率。但在进一步提高进程的并发性时，人们发现进程切换开销占的比重越来越大，同时进程间通信的效率也受到限制
- 线程的引入正是为了简化进程间的通信，降低运行实体间切换的开销，提高进程内的并发程度
- 线程：有时称轻量级进程，进程中的一个运行实体，是一个CPU调度单位，资源的拥有者还是进程

线程模型



- 多线程进程：一个线程被阻塞时，可运行同一进程中的另一 线程

感受线程编程（Java）1

```
public class MyThread extends Thread {
    int count= 1, number;

    public MyThread(int num){
        number = num;
        System.out.println("创建线程 " + number);
    }

    public void run() {
        while(true) {
            System.out.println("线程 " + number + ":计数 " + count);
            if(++count== 6) return;
        }
    }

    public static void main(String args[]){
        for(int i=0;i<5; i++) new MyThread(i+1).start();
    }
}
```

感受线程编程（Java）2

```
D:\tmp1>java MyThread
创建线程 1
创建线程 2
线程 1:计数 1
线程 1:计数 2
创建线程 3
线程 1:计数 3
线程 1:计数 4
线程 2:计数 1
线程 1:计数 5
线程 3:计数 1
创建线程 4
线程 3:计数 2
线程 3:计数 3
线程 2:计数 2
线程 3:计数 4
线程 3:计数 5
线程 4:计数 1
线程 4:计数 2
线程 4:计数 3
线程 4:计数 4
线程 4:计数 5
创建线程 5
线程 2:计数 3
```

```
D:\tmp1>java MyThread
创建线程 1
创建线程 2
线程 1:计数 1
线程 2:计数 1
创建线程 3
线程 2:计数 2
线程 1:计数 2
线程 2:计数 3
线程 3:计数 1
创建线程 4
线程 3:计数 2
线程 2:计数 4
线程 1:计数 3
线程 2:计数 5
线程 3:计数 3
线程 4:计数 1
线程 4:计数 2
创建线程 5
线程 4:计数 3
线程 3:计数 4
线程 1:计数 4
线程 3:计数 5
线程 4:计数 4
```

理解线程

- 线程是进程的一个实体，可作为系统独立调度的基本单位
  - 有执行状态（状态转换：阻塞、就绪、运行）
  - 不运行时保存上下文:每个线程有自己的程序计数器、堆栈（局部变量）、状态等
- 调度：线程作为调度的基本单位，同进程中线程切换不引起进程切换，当不同进程的线程切换才引起进程切换
  - 线程间切换时，系统开销小、切换快
- 并发性：一个进程内的多个线程可并发
- 拥有资源：线程仅拥有隶属进程的资源；进程是拥有资源的独立单位
  - 进程的多个线程都在该进程的地址空间活动，可共享该进程的资源

使用线程的优点和应用场合

- 线程间切换时，系统开销小、切换快
- 线程间共享进程资源，通信方便

- 适宜的应用场合
  - 非计算密集型任务，有比较多的IO操作
    - ➔如：接收数据、处理数据、备份数据可考虑设计3个控制线程，不会因为某个线程阻塞，导致整个进程被阻塞
  - 有多核CPU
  - 进程间通信代价太大
  - 需要并发

提醒：线程的数量也不是越多越好，合适最好

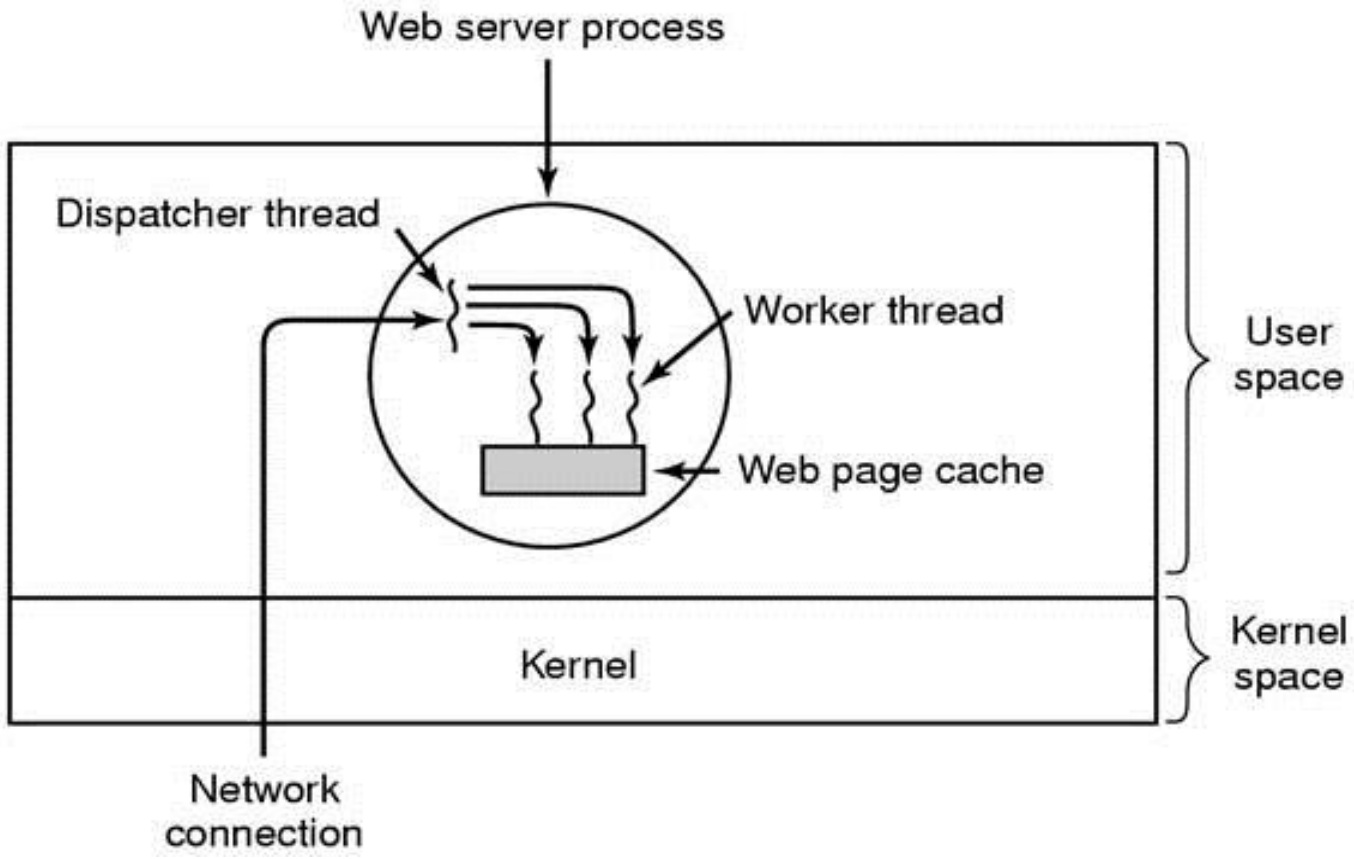
使用例：客户程序中的多线程

- 以web浏览器为例，web文档由HTML文件组成，包含文本文件、图像组、图标等，须建立TCP/IP的连接，建立连接和读取数据都可能导致阻塞

- 以多线程的方式开发浏览器，激活多个线程，每个线程都与服务器建立独立连接获取数据，每个线程负责获取页面的相应部分

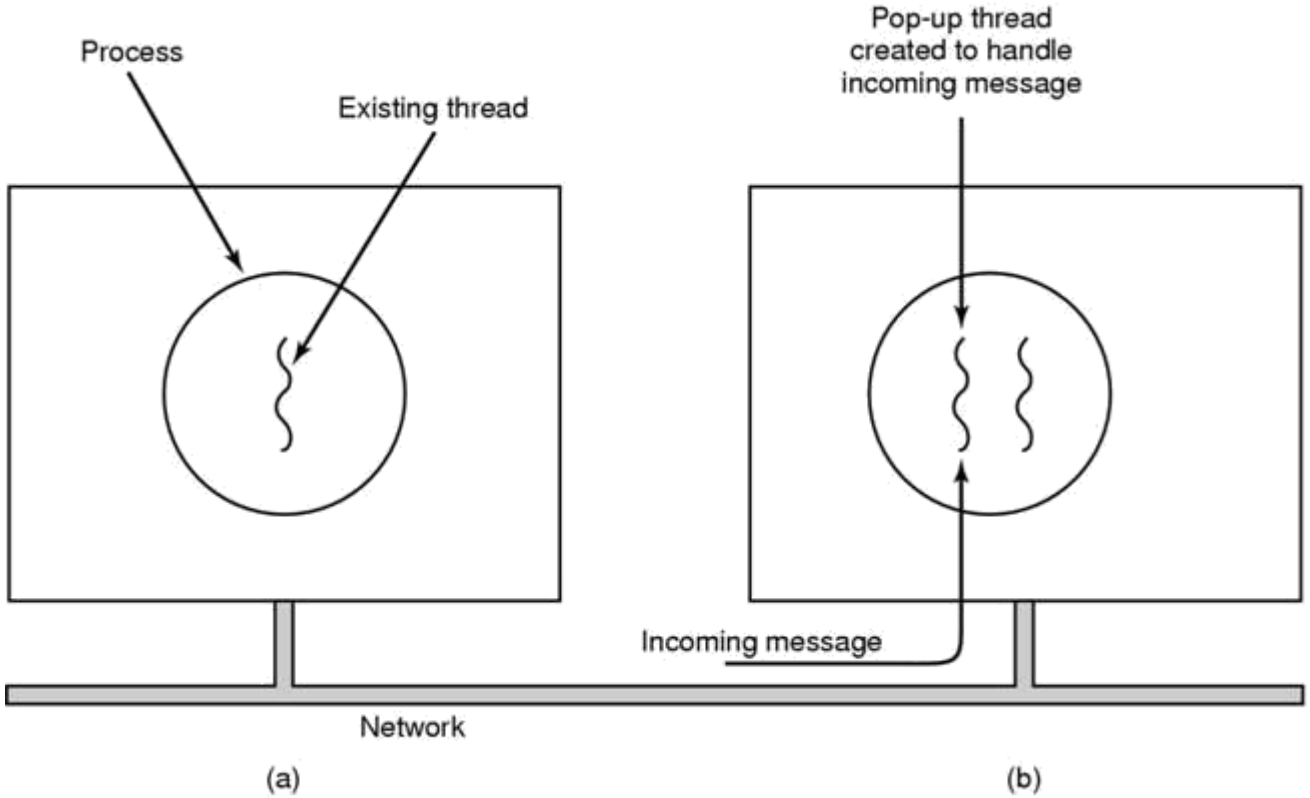
使用例：Server侧的多线程

- 如多线程的Web服务器



远程过程调用RPC中的线程

- **作用：负责接收消息。**



（a）消息到达前　　（b）消息到达后

提醒：没有免费的午餐

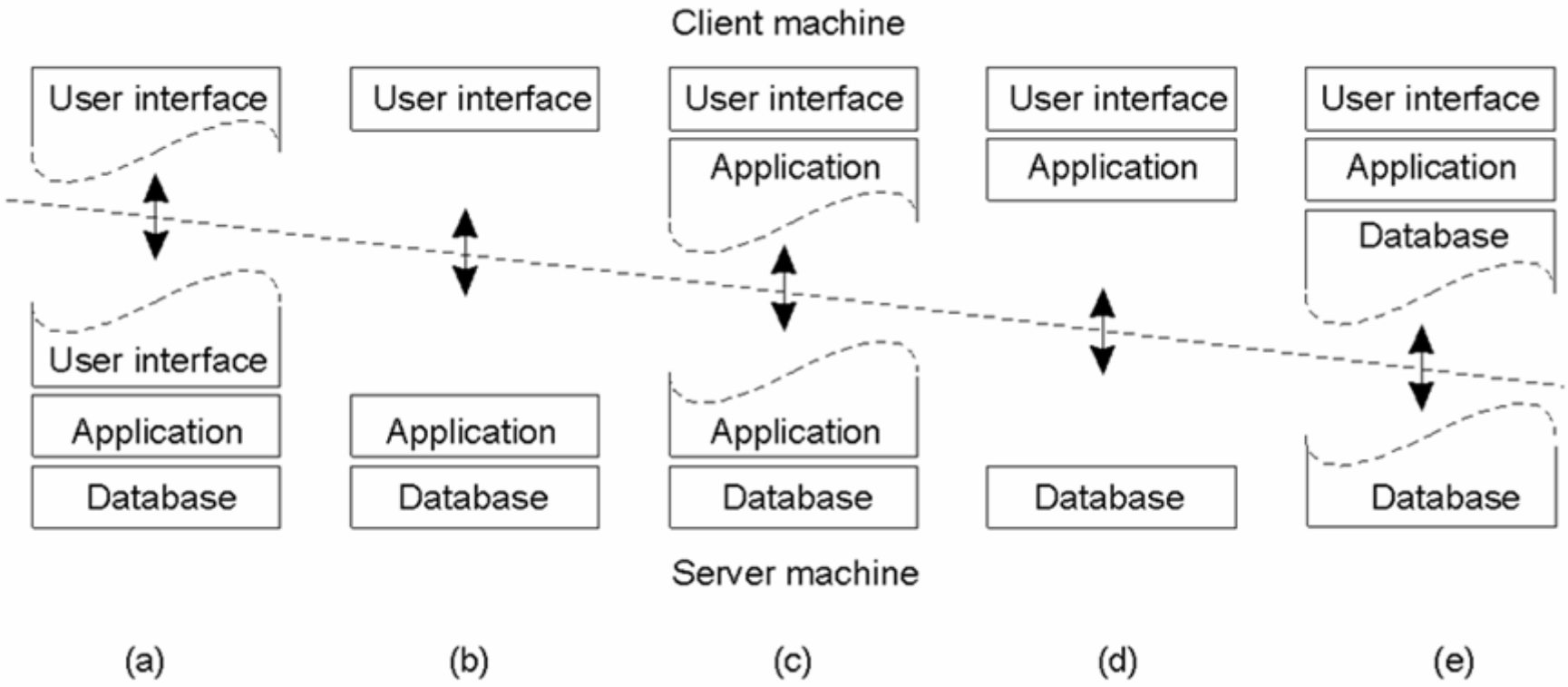
- 由于线程并不像进程那样彼此隔离，在单个进程中共享资源的并发控制问题由应用程序开发者负责解决
- 因此多线程应用程序的开发需要付出更多的努力
- 设计要合理，实现要简单

进程主要内容

- 进程
- 线程
- 客户
- 服务器

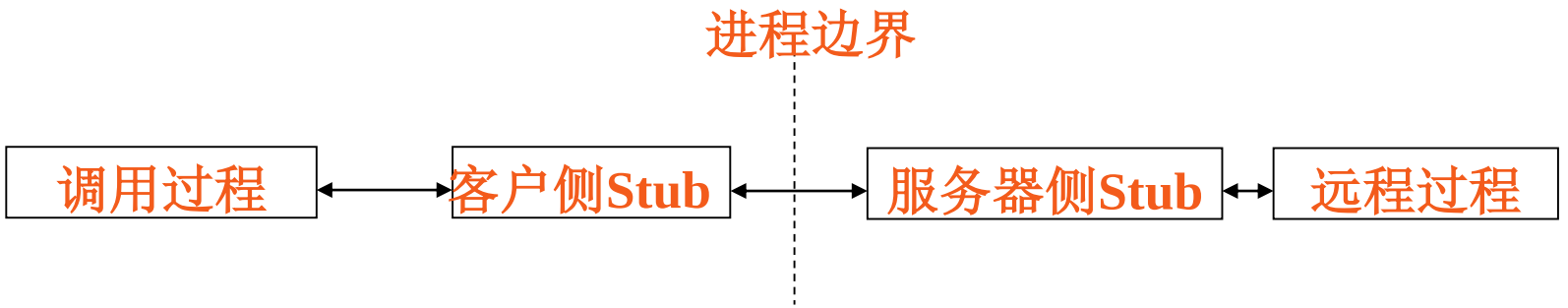
客户程序的设计

- **客户程序：根据需要可有多种设计思路**



客户程序对分布透明性的支持

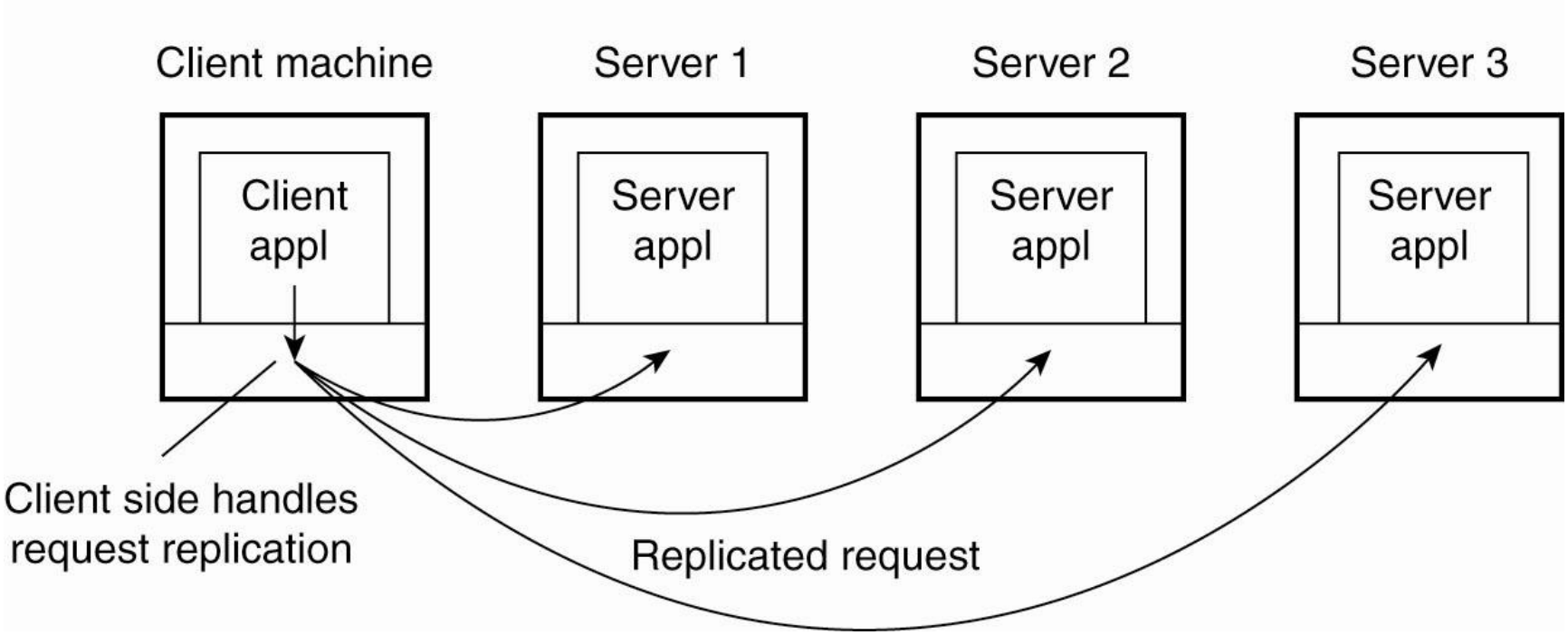
- 访问透明性：用相同的操作访问本地资源和远程资源
  - 客户、服务器存根程序(stub)：代理程序
  - 隐藏客户和服务器主机之间的差异和通信



- 位置透明性、移动透明性、重置透明性
  - 命名系统：全局逻辑名字
  - 重新绑定机制：当服务器改变位置后，通知客户侧重新自动绑定

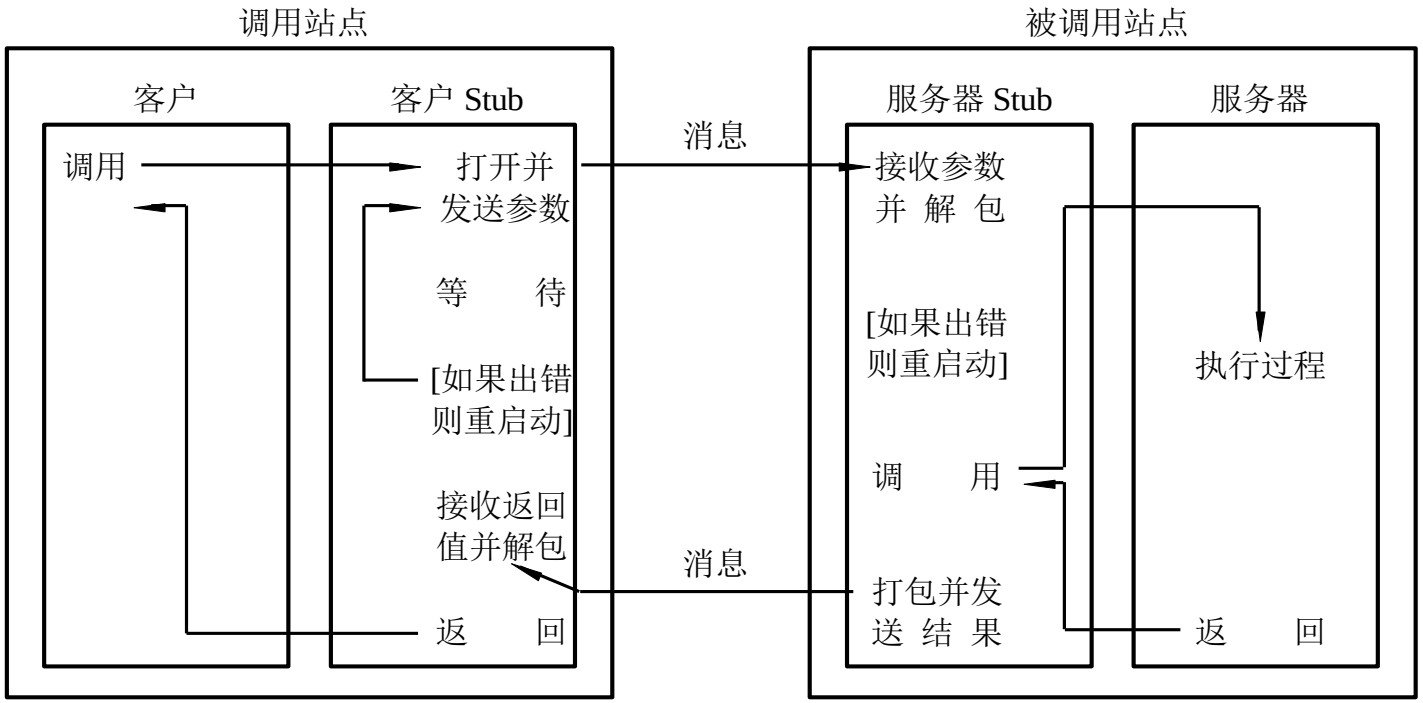
客户程序对分布透明性的支持(2)

- 复制透明性：使用客户端实现服务器复制透明性



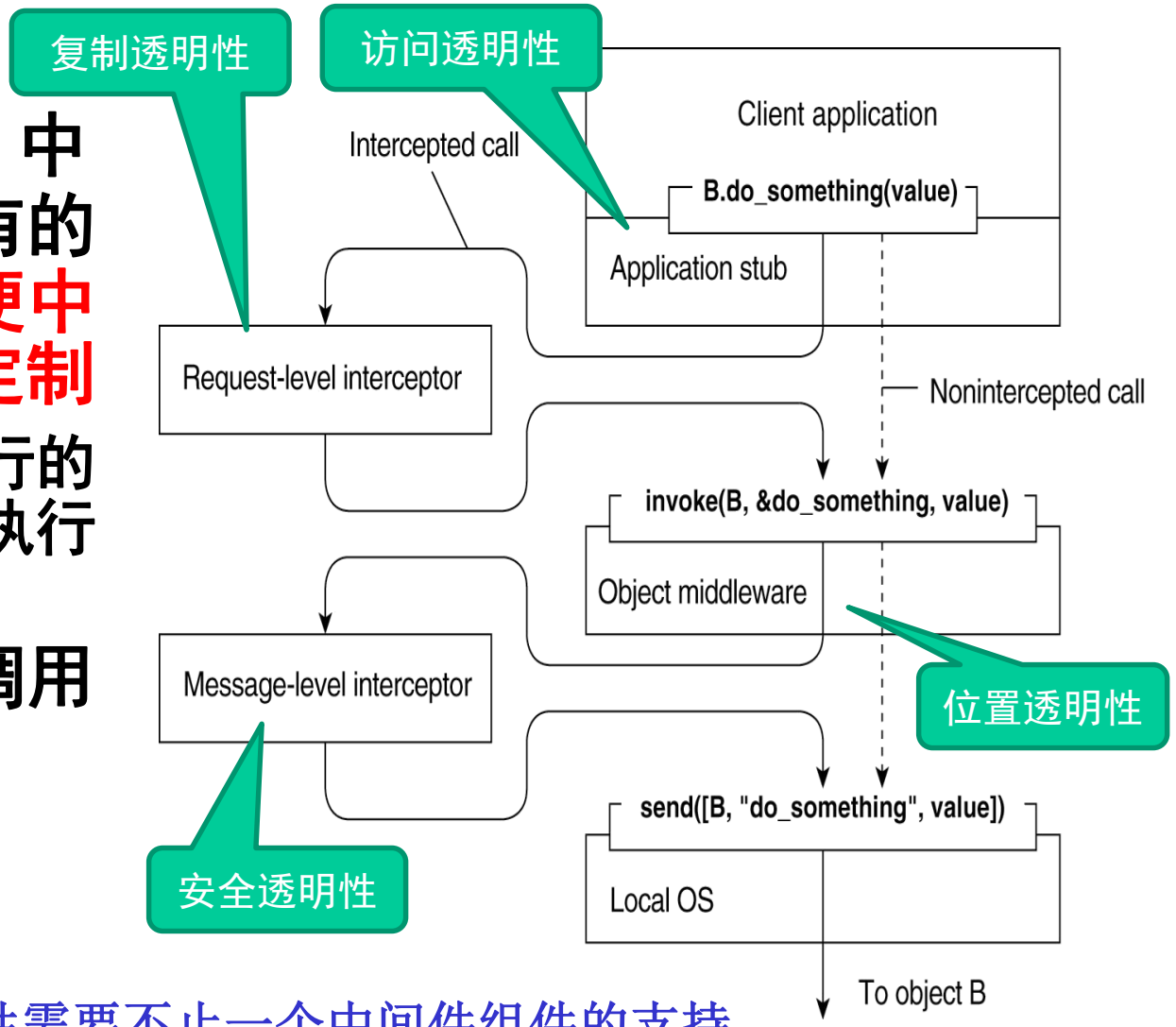
客户程序对分布透明性的支持(3)

- 故障透明性
  - 重新连接：通信失败时，重试
  - 本地数据缓存：例，脱机方式



拦截器与客户侧透明性支持例

- 拦截器（Interceptor）：中间件中大多具有的一种组件，方便中间件的配置和定制
  - 可中断正常执行的控制流，插入执行其他代码
- 例：远程对象调用
  - 请求级拦截器
  - 消息级拦截器



提醒：大部分透明性需要不止一个中间件组件的支持

进程主要内容

- 进程
- 线程
- 客户
- 服务器

服务器

- 服务器是实现特定服务的进程，这些服务是为一组客户提供的
- 本质上，每个服务器的工作方式都是一样的：等待来自客户的请求，随后负责处理该请求，然后等待下一个请求
- 一般情况下，服务器同时为多个客户提供服务，因此，同一时刻可能会有多个请求到来

服务器的组织结构

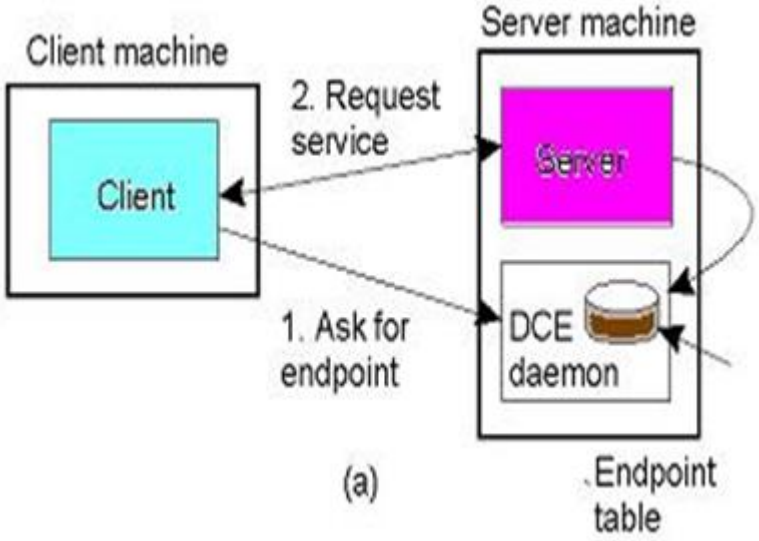
- 迭代服务器：自己处理请求，并在必要的情况下将响应返回给发出请求的客户
- 并发服务器：自己不处理请求，而是将请求传递给某个独立的线程或者其他进程来处理，自身立即返回并等待下一个输入的请求
  - 如：多线程服务器
  - 再如：每收到一个请求都派生出一个新的进程来进行处理
  - 由处理请求的线程或者进程负责向发出该请求的客户端返回响应

客户如何联系服务器？

- 客户如何知道服务器的IP地址？
  - 众所周知IP地址、 DNS、命名服务
- 客户总是需要向服务器所在的某个端口（Port）发送请求，而服务器应该在这个端口监听请求
- 客户如何知道某个服务所对应的端口？
  - 方法1：为已知服务分配一个众所周知的端口
    - ➔FTP 21
    - ➔HTTP 80
  - 方法2：服务器的端口被本地操作系统动态分配，客户必须找到该端口
    - ➔怎么找到？

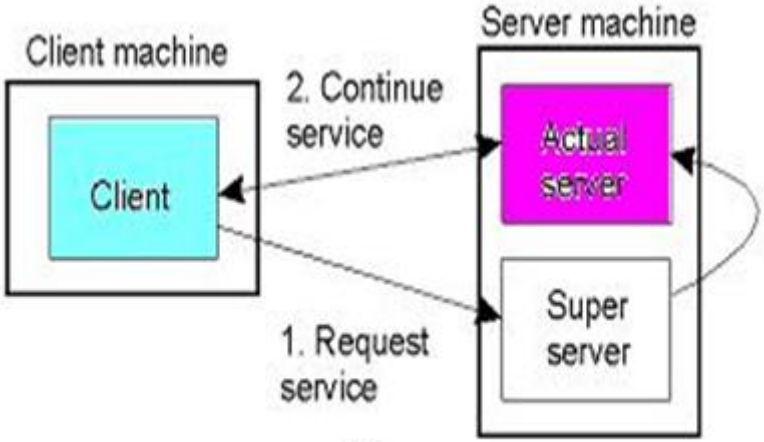
客户到服务器的两种绑定策略

- 方法1：在运行服务器的每台机器上运行一个特殊的守护进程 daemon
- 该进程负责跟踪位于同一台机器上的每一个服务进程使用的当前端点
- 该进程监听一个已知的端口，客户通过这个端口与该守护进程进行联系，得到服务器的端口



客户到服务器的两种绑定策略（2）

- 方法2：使用超级服务器
  - 如Unix中的inetd守护程序，监听许多Internet服务的已知端口
  - 当收到请求的时候，它派生出一个进程以对该请求进行进一步处理，这个派生出的进程在处理完毕后自动退出运行



(b)

客户对服务器工作的中断方法

- 方法1：让用户强行退出客户应用程序（这将中断与服务器的连接）然后马上重新启动客户程序，就好像什么事情都没有发生一样。
  - 看起来比较笨，但在互联网环境下工作得挺好
- 方法2：使用带外数据
  - 利用单独另外的控制端点
  - 利用同一链路，如，TCP中发送urgent data

服务器的状态问题

- 永久状态和会话状态
  - 永久状态记录的通常是数据库数据，如客户信息
  - 会话状态是服务器与一个客户间一次会话中涉及的信息、数据
    - ➔会话(Session) 是通信双方从开始通信到通信结束期间的一个上下文（Context）

- 服务器（进程）的状态问题通常关注的是**会话状态**

有状态服务器

- 有状态服务器：服务器保存会话状态，同一会话的请求之间有关联，即请求的响应结果与该会话之前的请求有关
  - NFS文件服务器：存储文件使用表（客户，文件，可更新否）；服务器重启时：需要恢复故障前状态
  - 网络游戏服务器：在服务端维护每个连接的状态信息，服务端在接收到每个连接发送的请求时，可以从本地存储的信息来重现上下文关系
- 优点：方便建立与客户间的会话关系，支持有状态的服务
- 缺点：
  - 服务端保存大量数据，增加服务端压力
  - 客户端请求依赖服务端，同一会话的多次请求必须访问同一台服务器
  - 服务端保存用户状态，难以进行水平扩展（增加服务器数量）
  - 服务器发生故障时，状态恢复比较困难

无状态服务器

- 无状态服务器：服务器不存储会话状态，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），要么是自身所保存的、并且可以被所有请求所使用的公共信息
  - 例如，Web服务器：每次HTTP请求和以前的请求没有直接关联，仅仅对这次的HTTP请求作出响应，如获取一个文件

    ➔HTTP是无状态协议
- 优点
  - 客户端请求不依赖服务端的信息，任何多次请求不需要必须访问到同一台服务器，服务器副本之间不需要进行状态同步
  - 服务器发生故障时，不需要进行状态恢复
  - 服务端可以方便地进行水平扩展
- 缺点：支持有状态的服务比较困难

用无状态服务器实现有状态服务

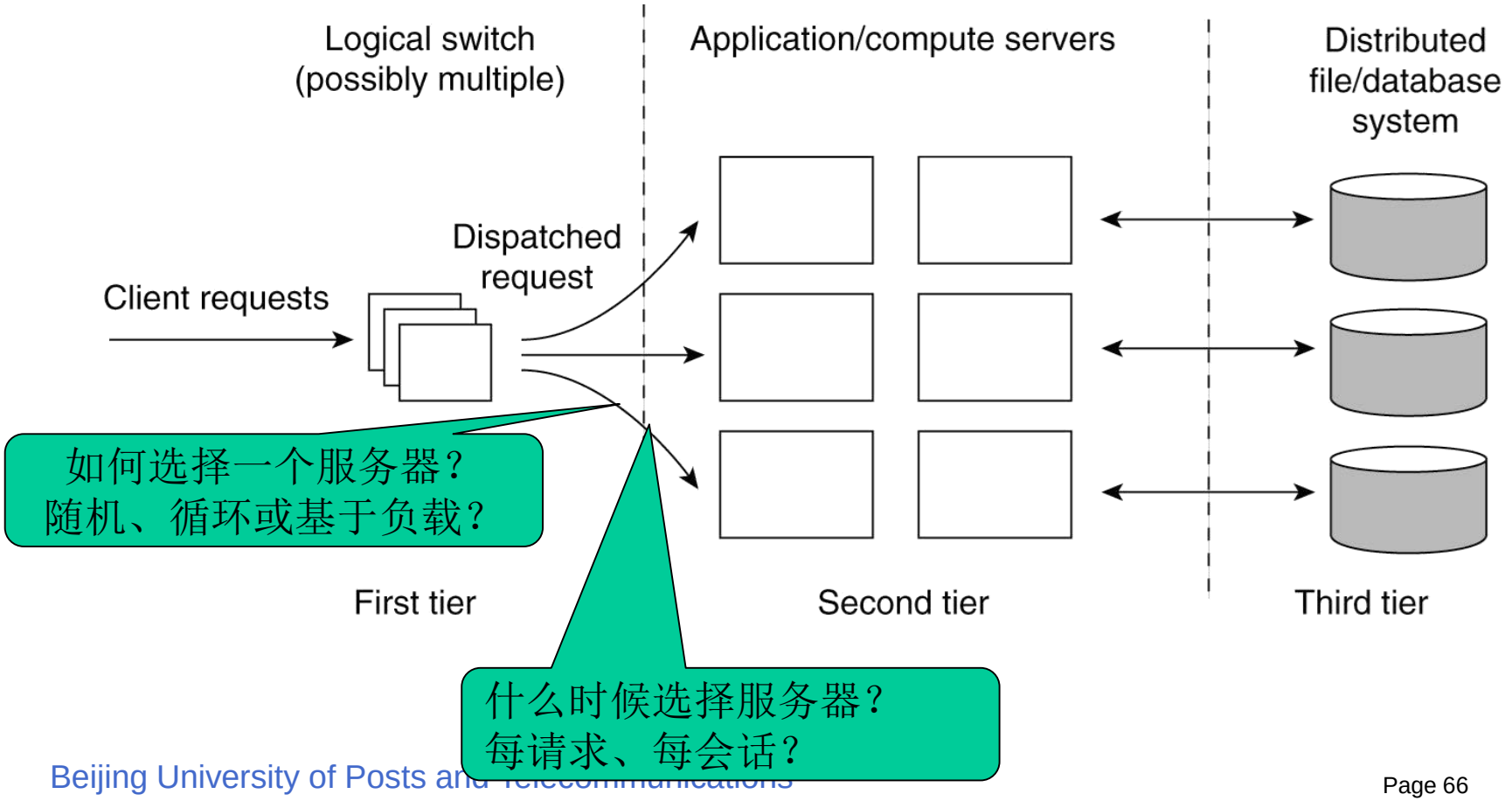
- 在某些情况下，服务器需要保留客户的活动记录
  - Web服务器可以将客户引导到该客户最常浏览的页面
- 实现方式例：
  - 使用Cookie：是一小段数据，包含对服务器有用的针对特定客户的信息，存储在浏览器中。客户每次访问该服务器时，相关Cookie将与请求一起发送给服务器
    - → 如保存用户的登录信息，会话ID。对用户是透明的
    - → 缺点：违反隐私权，有安全性隐患（仿冒ID）
    - → 如果用户不允许浏览器保存cookie？

服务器集群

- 是一组通过网络连接的机器，每台机器运行一个或多个服务器
  - 这里讨论的服务器集群是指经局域网连接的机器，能提供高带宽和低延迟
  - 云计算技术特别适合搭建服务器集群

常见组织形式

- 服务器集群逻辑上常由三层组成
  - 具体组织方式需要具体问题具体分析



进程总结

- 分布式系统中，进程是基本部分，它们协作实现了分布式系统的功能，也构成了不同机器间通信的基础
- 使用多线程可以构建更高效的服务器，而且可以有效地支持并发
- 客户进程一般实现用户接口，通过隐藏与服务器通信的细节，可获得更好的分布透明性
- 服务器可以是迭代的也可以是并发的，可以实现一种服务也可以实现多种服务，可以是无状态的也可以是有状态的
- 很多服务器组织成集群，通常需隐藏集群内部细节，使用单访问点将请求消息转发给服务器