# Software Architecture

- principle and practice

---

# 第五讲　层次结构技术分析

- 异步处理：进程中若有两个或多个任务，它们之间的处理顺序没有规定，则每个任务可以由一个线程处理

## 二、服务器缓冲技术

### 1.无状态信息(Stateless)服务器

- 一个服务器称为无状态信息的，当一个客户发送的一个请求给服务器时，服务器执行请求，返回结果，然后删除该请求的有关所有信息。

### 2.有状态信息(State)服务器

- **通常集中式系统中，对于活动的进程，系统往往保留状态信息，而对分布式系统来说，是否也可以借鉴?**
- **有状态信息服务器(Stateful information server):文件服务器拥有打开、关闭、读、写等操作，当一个文件被打开后，服务器将保存是哪一个客户打开的，打开的是哪一个文件，并生成一个文件描述符(file discroptor)。一个客户如果已经打开了一个文件，那么它的后续操作，只要给出文件描述符及有关的参数即可，服务器收到请求后，根据文件描述符就知道哪一个文件。状态信息(Stateful information)是一个映射表，文件描述符映射到它的文件。**

### 3.无状态服务器与有状态服务器的比较

- **对于无状态服务器，每一个请求都要求是完备的、独立的，包含完整的文件名、偏移量，以便服务器的工作，请求消息长度增加。**
- **无状态信息服务器具有更高的容错性。**
- **Open/Close调用没有必要，从而减少传送消息的次数**
- **服务器不会因为保留状态信息而浪费空间，因为某一时刻当有大量客户要打开大量文件时，表将填满而新文件不能打开，客户的请求得不到服务，正确的程序将不能正确地工作，无状态信息服务器将消除这个问题。**
- **客户机崩溃也不会产生问题。**
- **文件加锁是不可能的，因为无状态可登记，这种情况下，无状态信息服务器不得不使用加锁服务器。**
- **对于有状态信息服务器，Read/Write消息中不必包含文件名，它们的长度将缩短，从而占用更少的网络带宽;**
- **由于打开的文件在内存中，读、写操作速度更快，性能得到提高;**
- **因为更经常的是读操作，所以可以事先成块读到内存而减少延时;**
- **同一有效性容易实现，如果客户因超时重传同样的请求，服务器有两个接收的情况下，由于内存中保存了状态信息，很容易通过比较每个消息的序号而发现;**
- **文件加锁是可能的。**
- **A comparison of stateless and sateful server**

### 4.无状态服务器与有状态服务器的比较总结

| advantages of stateless server | advantages of stateful server |
|---|---|
| Fault tolerance | Shorter request message |
| No pen/close calls needed | Better performance |
| No server space wasted on table | Readahead possible |
| No limits on numbers of opened file | Idempotency easier |
| No problem if a client crashes | FIle locking possible |

## 三、N层结构的特性

- 层次结构是解决复杂问题时最常用的软件结构，
- 1.层次结构设计
  - 层次
    - 完成若干功能或服务的模块，这些功能具有独立性;
    - 一个层次是一个服务的提供者;
    - 一个层次还可以进一步分层;
  - 层次间的单向依赖关系
    - 同一系统的层次之间构成单向的依赖关系，一个层次依赖于较低的层次。这种依赖关系是通过接口实现的。
  - 层次的隐藏性
    - 层次间的单向依赖关系使得每个层次具有隐藏性。一个层次可以隐藏其内部的实现细节，向上提供一个一致的服务。使高层次可以不必了解低层次的细节，如物理特性、存储方式、位置等。
  - 分层的原则
    - 第n层存在的必要性：它对n-1进一步完善和扩充，并提供n+1的服务接口更简单可靠。
    - 从这个意义上说，层次的隐藏性是不完全的，如果第n层中的一项服务已经足够完善，则该服务就不必在n+1层中继续存在，这样，第n+2层在需要该服务时可直接使用第n层中的服务。
- 2.软件系统的层次结构
  - 研究表明，任何软件的完整结构都具有层次关系
    - 硬件基础层：这是软件得以运行的物质基础，它包括：处理器、存储器、时钟、中断及其控制、I/O端口、I/O通道、快速缓存、DMA等。软件是针对特定硬件的构成而设计的，反映对硬件的支持的需要，即硬件发生变化后，原则上软件也需要做出相应的为变更。

- 软化的硬件层：在对硬件结构和性能进行描述的基础上，实现硬件的操作和控制描述，这就是软化的硬件层。在该层次上，处理器被软化为状态和指令的集合，中断被描述为状态和中断服务的集合等。该层是软件构成的基础，其程序设计工具主要是汇编语言(assemble language)。
- 基础控制描述层：主要包括高级语言的所支持的程序控制和数据描述。程序控制的概念有：顺序、条件、选择、循环、变量、参数、生存期、程序、过程/函数等，数据描述的概念有：数组、结构/记录、队列、树、图、指针等，以及面向对象中的类、对象、继承、多态、重载等。该层次的工具是程序设计语言、结构化或面向对象的分析设计。
- 资源和管理层：在基础控制要描述层建立的一切对象和数据都需要在操作系统的协调和控制下才能实际地实现其设计的作用和功能。该层提供了基于操作系统结构的任务管理、消息处理、系统输入/输出控制，其他系统级别的资源和功能服务。该层的某些服务的定义在基础控制描述层中，但功能的实现是建立在操作系统管理层的。
- 系统结构模型层：我们知道，软件体系结构是软件的“高层(high level)结构”，该层包括的概念如：解释器、管道/过滤器，C/S、黑板等
- 应用层：这是纯粹从应用领域出发所建立的系统结构概念，该层包括的概念如，企业管理、公文处理、控制系统、CAD系统等。

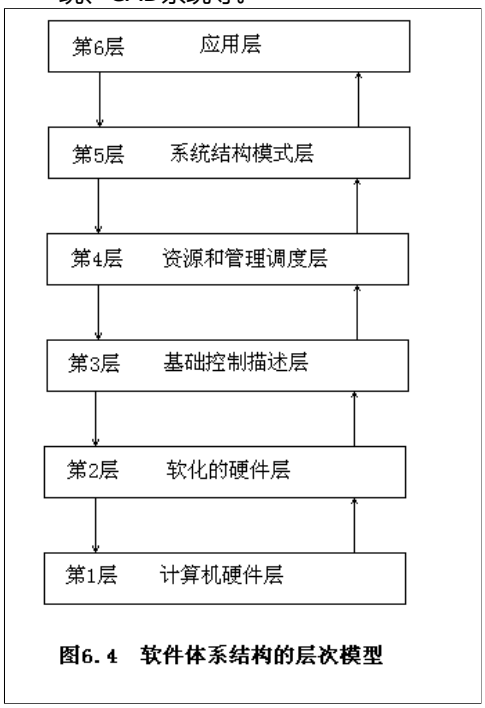


图6.4 软件体系结构的层次模型

- 3.N层结构
  - C/S或B/S结构称为2层结构，我们把层次结构中3层或3层以上统称为N层结构。

## 四、N层结构的实现

- 1.一个产品的软件体系结构风格如果采用N层结构。则可能按下面的步骤实施
  - 定义为合适的分层而采取的抽象标准：在软件开发中，根据距硬件接近或距应用接近的程度建立分层，比如某个应用可能建立如下的分层：用户界面、特定功能模块、公共服务、操作系统接口、操作系统、硬件层。
  - 抽象标准决定模型层次的数目：分层的原则是对于第n+1层来说，如果第n层提供的功能不会比第n - 1层简单，则第n层就不必存在。
  - 给每个层次命名和分配任务：在层次结构中最高层的任务就是整个系统从用户出发的任务。如果采用自底向上的设计方法，较高层次是建立在较低层次之上的，这要求对系统具有丰富的经验和敏锐的洞察力，以便在确定高层前找到低层次的合适抽象。
  - 规范服务：层次之间要严格分开，确保没有部件会跨越两层以上，层J函数的参数、返回值和错误类型者应限定在程序语言的类型、层J定义的类型或从共享数据模块中引用的类型。
  - 为每个层次定义接口：在层J中定义一套良好的供层J + 1层使用的服务接口(interface),使用层J + 1看不到层J对接口对应的服务的实现细节，
  - 设计错误处理策略：尽可能把错误处理在更低层次上，避免高陷入更多的错误。
- 2.Window NT的层次结构
  - 系统服务层
  - 资源管理层
  - 内核
  - 硬件抽象层
  - 硬件层

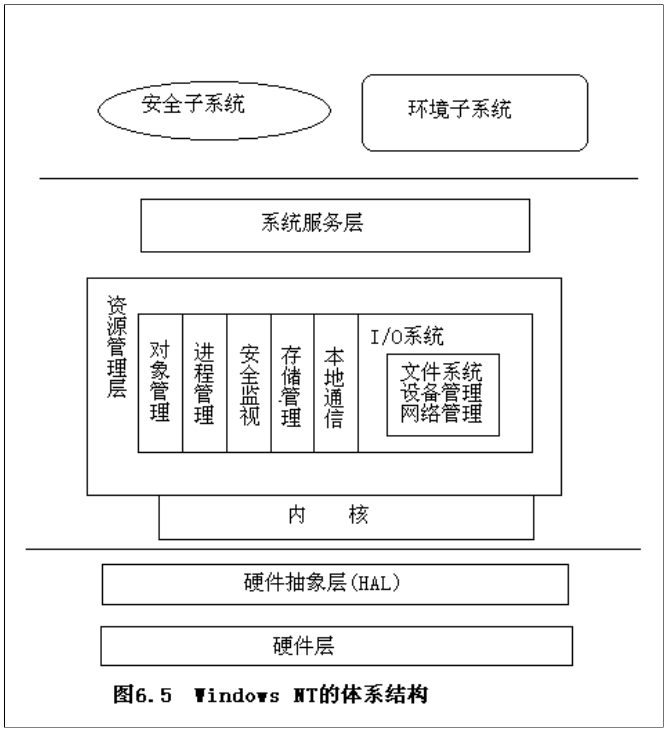


图6.5　Windows NT的体系结构

## 五、N层结构的优缺点

### 优点

- 层次的利用性：如果层次中很好地体现了抽象、并且具有定义良好、文档化的接口，那么该层能在多个环境中使用。
- 对标准化的支持：清晰定义并广泛接受的抽象层次能够促进实现标准化的任务和接口开发，相同接口的不同实现能够互换使用，这样就允许在不同的层使用来自不同组织的产品。
- 依赖性本地化：层次的标准接口通常把代码的变化的影响限制在其所在的层次上，支持了系统的可移植性和可测试性。
- 可替换性：独立层次的实现能够轻易地被功能相同的模块替换。如果层次之间的联系在代码中是固定的，那么联系能够根据新层次实现的名称来更新。比如，硬件的可替换性，新的硬件I/O设备通过安装正确的驱动程序就能够使用，互操作性不影响高层次，高层次的接口不要改变，可以像以前一样继续向低层请求服务。
- 位置透明性：通常低层次把服务所在的网络物理位置隐藏起来，使高层次的请求可以不关心其服务是在本地或是在远程。

### 缺点

- 改变行为的连锁效应：当某个层次的构成和行为发生变化时会生产生严重的连锁效应，在维护升级时，如果必须在许多层次上做大量的工作，那么层次结构将变成一种缺点。
- 低效率：分层结构通常要比单层结构的效率低。如果高层服务严重地依赖于底层服务，那么必须穿越许多中间层进行数据的拷贝。
- 分层是有限制的：并不是所有的产品都可以分层，同一产品需要分成几个层次没有统一的标准。

## 六、一个用于构造分布式系统的层次结构设计

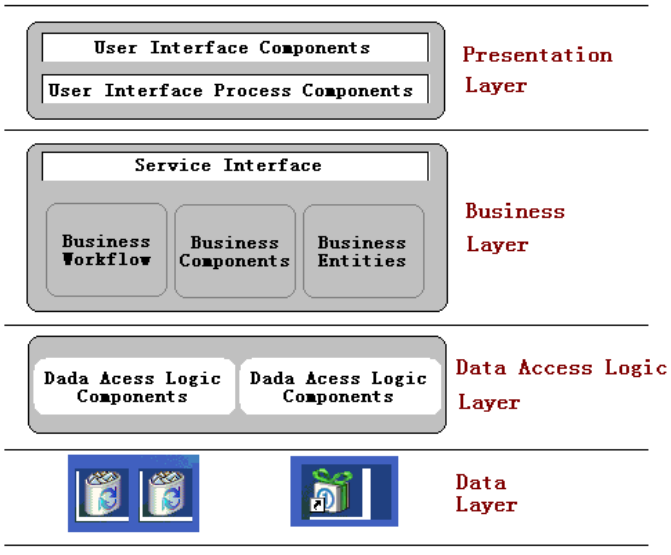
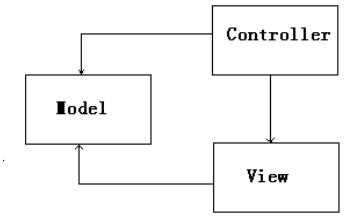


图6.6　层次结构

- 1.表示层 - - 用户界面技术
  - 划分UIC和UIPC，进一步抽象，提高代码的重用性和降低模块的耦合度
  - UI - - 用户和应用进行交互的接口
    - 提供的功能
      - 输入方面
        - 辅助用户输入，提供各种提示和帮助，在输入的同时会有一些校验，如日期等
        - 响应用户操作所触发的各种事件（可能会展现出其他的UI）。
        - 限制用户的输入，在数据改变的时候，可能会有一些相关联的操作（如，单价的改变，影响到总价）。
        - 处理一些特殊的操作（如drap-drop,剪贴板等）
      - 输出方面
        - 格式化数据（如金额、日期等）
        - 特殊显示（坏帐号用特殊颜色标识出来）
        - 将一些编码转换成有意义的名称
      - 个性化（Web页面，Winfrom(dialog,MDI,SDI)）
      - 其他(状态、分页显示查询结果等)
    - UI与UI之间的调用的代码不要写在UIC中，由UIP处理，提高重用性
    - UIPC - - 处理用户UI的流程控制
      - MVC模型
        -
        
        
        图6.7 MVC模型
        -
        - View。用户操作界面
        - Model。内部数据结构，状态数据
        - Controller。控制流程，UIPC的核心
      - 什么是UIPC
        - 根据状态改变决定使用哪一个UI。
        - 应用场景
          - 有些UI之间的相互作用，存在明确的处理顺序（向导界面的上一、下一等，购物网站的浏览、选择并加入购物车到收银台结帐）
          - 这些类型的界面操作的特点：
            - 用户操作流程可以用一张流程（导航）图来描述）
            - 导航图上每一个节点是一个用户界面（窗口、页面）
            - 界面之间的跳转由用户操作触发的
        - 处理这种流程的控制器称为UIP
      - UIPC的好处
        - 隔离UI与业务逻辑层
        - 对流程中的UI进行管理
        - 提供状态保存和传递机制
      - 状态保存
        - Server状态和Client状态，状态有效期?
      - Smart Client
        - 智能安装和版本更新
        - Connected。选择一个合适的有效的Service
        - 能够利用本地资源（CPU、HD等）
        - 离线能力。在不连接网络上工作，连线时提交数据
- 2.业务逻辑层 - - 应用系统的核心
  - Business Component
    - 含义
      - 实现业务规则及执行业务工作的组件，负责发起事务，实现业务功能
    - 特点
      - 由用户处理层的UIPC、服务接口、以及其他业务组件调用，包含一些业务数据和操作，以及复杂的数据结构
      - 是事务的发起者，参与事务的提交
      - 必须验证输入和输出
      - 通过调用数据层组件获取数据或修改应用数据
    - 设计
      - PIPELine Parttern.顺序规定

        - Event Parttern.顺序不固定
- Business Workflow
    - 含义
        - 具有各种不同功能的活动相连的一组有相互绕道而行关系的任务。
        - 业务流程有起点和终点，而且它们都是可重复的
        - 由多个Business Components组成，有一定的顺序。
    - 特点
        - 迅速实现商业规则和商业目标的改变的能力
        - 将每一步业务操作、资源管理以及流程独立分离
        - 以前后一致的方式定义、改变和实现业务流程
    - 种类
        - 基本于人的业务流程：他要完成、批准、执行的文档
        - 基于规则的自动化流程：应用程度彼此连接，在无人干预的情况下进行合作
    - 实现
        - 流程引擎 - - Business Workflow的核心
            - 实现业务流程，同时管理活动的启动和终止，或业务功能
        - 资源管理
            - 使实现商业功能或活动所必须的资源具有可用性
        - 调度程序
            - 资源可用性的限制，商业功能经常受时间约束，需要调度程序以使时间约束和资源可用性相匹配
        - 审计管理
            - 关键组件，跟踪谁操作什么
        - 安全管理
            - 资格授权
- Business interface
    - 含义
        - 服务接口是一组软件实体，为实现处理映射和转换服务的外观组件（facade），把业务逻辑表现为服务，为服务提供进入点。
    - 作用
        - 提供业务处理的调用点
        - 实现缓冲、映射、以及简单的格式和结构转换
        - 不实现业务逻辑
        - 进行信息安全控制：有的需要安全身份验证
        - 分隔内部系统的实现
            - 对内部实现进行更新时，不需要变更服务接口
            - 需要验证传入的消息
    - 特点
        - 将服务接口视为应用程序的信任界限
        - 同一功能的服务发布多种服务接口，不同接口执行不同的服务等级义协议（SLA）
        - 尽可能提高与其它平台和服务的互操作性
    - 实现
        - 服务接口使得使用者和提供者之间能够交换信息，负责实现通信时的所有细节
        - 网络协议
            - 应该封闭使用者和通信所使用的网络协议（如，服务由TCP/IP上的http向使用者提供，则服务接口可以实现为ASP.NET组件，发布URL，http请求、响应和返回等）
        - 数据格式
            - 负责对使用者的数据格式和服务所期望的数据格式的转换
        - 安全性：信任边界，敏感操作授权使用
        - 服务等级协议（SLA）：Service Level Agreement,服务接口缓冲、缩短响应时间、节省网络传输等，负载平衡功能、容错技术等
    - 方法
        - XML　Web　服务
        - 消息队列方式（MQ）
    - 优点
        - 接口与应用逻辑分离（重用、维护）
        - 部署灵活（代码与服务分离）
    - 缺点
        - 接口粒度设计
        - 增加在更改服务所需的工作量
        - 增加复杂性和性能开销
- Business Entities （BE）
    - 含义
        - 应用程序的逻辑可能在设计中需要考虑多种数据格式，UI层的数据与数据库中的数据格式、外部服务提供的数据格式等可以不同。BE提供了一个中间层。

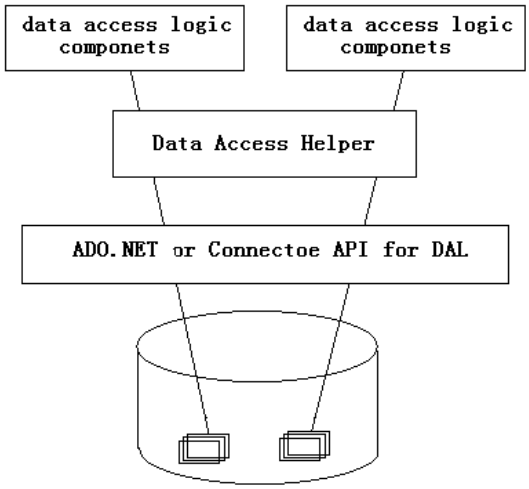
        - 作用
            - 将显示数据与实际存储隔离，保证业务实体的独立性，提高重用性
            - 业务实体一般是在应用程序中内部使用
            - 不同业务其数据格式不同
        - XML、dataset、datareader
- 3.数据逻辑访问层(DAL)
    -


图6.8 DAL
    - 好处
        - 增加代码重用性，可以被业务逻辑层在多个地方反复引用
        - 尽可能消除业务逻辑层对数据源的依赖（由于数据源改变造成的影响极小化），DAL可以通过配置文件进行改变（如Oracle改变为MS SQL Server），隐藏了数据操作的细节
        - 需要一个Helper来帮助完成数据操作，管理连接、缓冲等
        - Helper的作用
            - 为DAL提供通用的数据访问接口
            - 减少数据访问操作代码（简化执行SQL语句和调用存储过程的代码）
            - 进行数据连接管理
            - 在不同的数据源之间，可以提供统一的接口
- 4.数据层
    - 数据源：关系数据库、文件系统、Exchange Server、Web Storage等

## 七、作业

[返回]

==========