COMP3027J 课程

软件架构

软件架构风格概述

（调用/返回式）

邓永健

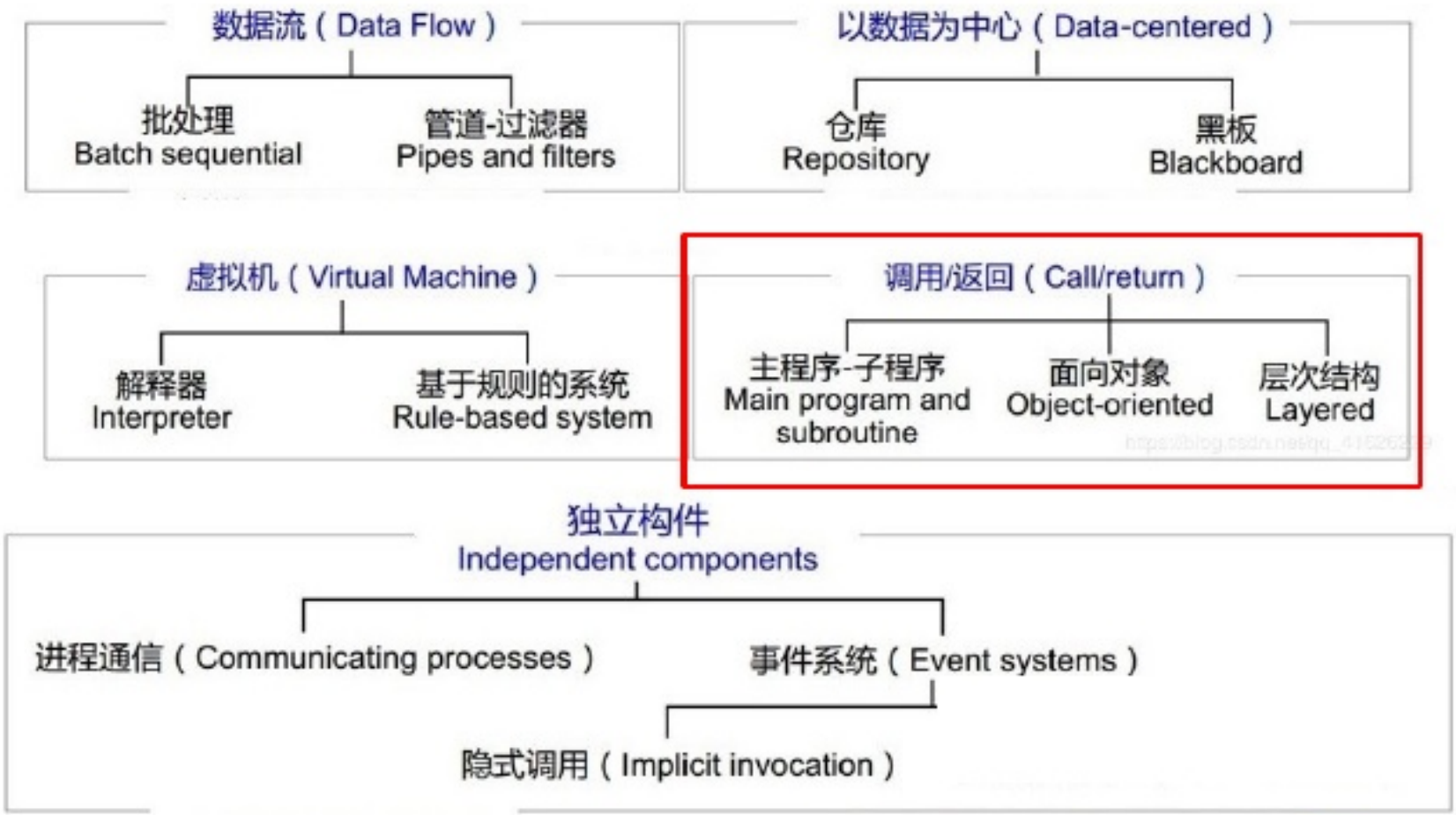
北京工业大学计算机学院

数据挖掘与安全实验室（DMS 实验室）

大纲

大纲

风格分类学


数据流（Data Flow）
批处理
Batch sequential
管道-过滤器
Pipes and filters
以数据为中心（Data-centered）
仓库
Repository
黑板
Blackboard
虚拟机（Virtual Machine）
解释器
Interpreter
基于规则的系统
Rule-based system
调用/返回（Call/return）
主程序-子程序
Main program and subroutine
面向对象
Object-oriented
层次结构
Layered
独立构件
Independent components
进程通信（Communicating processes）
事件系统（Event systems）
隐式调用（Implicit invocation）

调用/返回架构风格

主程序与子程序

- 经典编程范式：功能分解。

面向对象

- 信息（表示，功能）隐藏。

分层的

每一层都只能与其直接相邻的层进行通信。

C/S、B/S 等等。

调用/返回架构风格

主程序与子程序

单线程控制，分为几个处理步骤。

- 功能模块：将步骤整合为模块。

面向对象

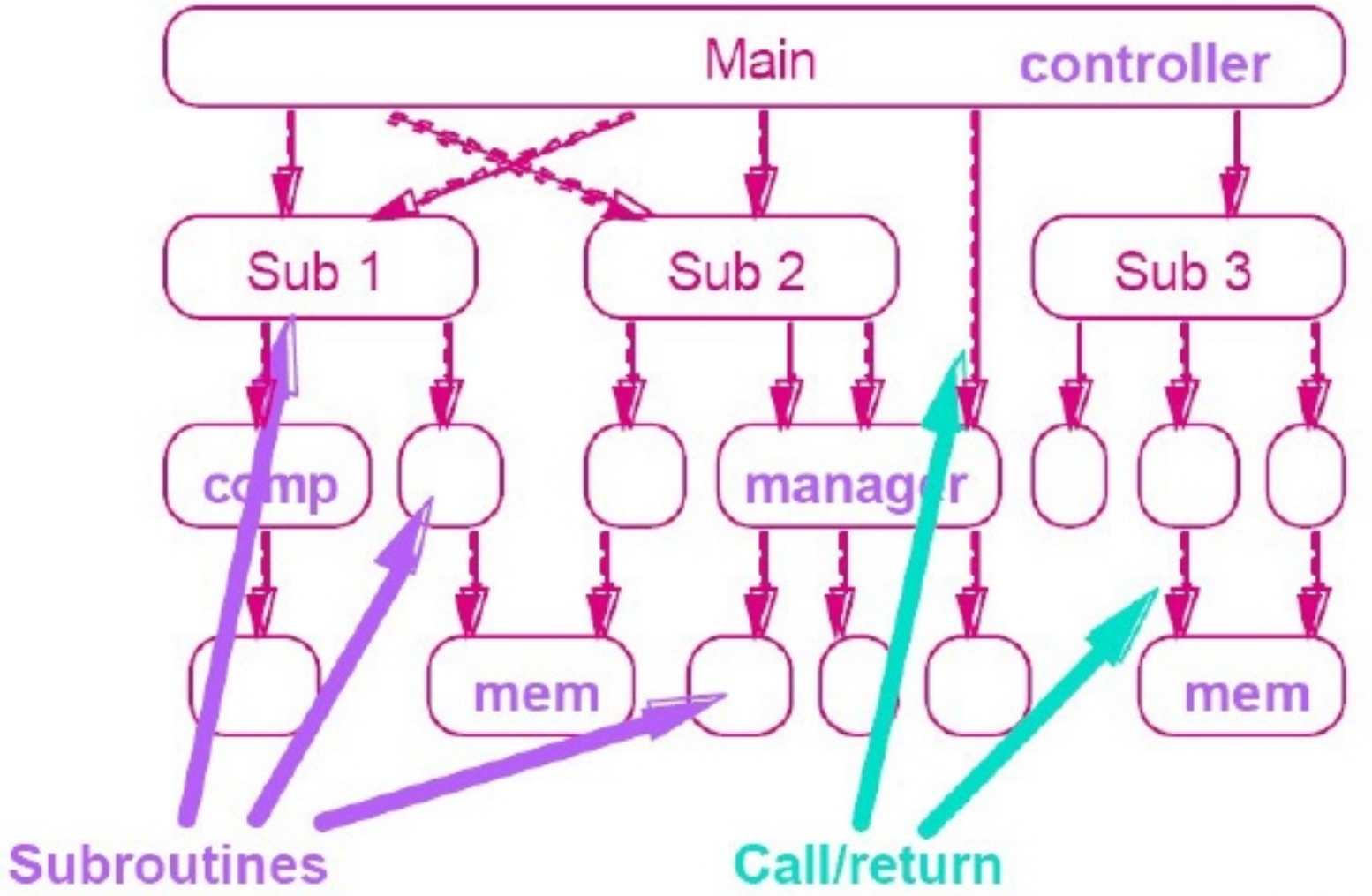
- 方法（动态绑定）、多态性（子类化）、复用（继承）。

对象在不同的进程/线程中处于活动状态（分布式对象）。

| 客户端/服务器，分层式风格

大纲

主程序与子程序


Main
controller
Sub 1
Sub 2
Sub 3
comp
manager
mem
mem
Subroutines
Call/return

主程序与子程序

问题：这种模式适用于那些计算能够通过一系列过程定义的层次结构来恰当地定义的应用程序。

许多编程语言都自然地支持定义嵌套的**过程**集合，并允许分层调用它们。这些语言通常允许将过程集合分组到模块中，从而引入了名称空间的局部性。执行环境通常在单个名称空间中提供单个控制线程。

主程序与子程序

解决方案：

- 系统模型：调用和定义层次结构，子系统通常通过模块化来定义。
- 组件：程序和显式可见的数据。
- 连接器：过程调用和显式数据共享。
- 控制结构：单线程。

模块分解

帕纳斯

- 隐藏秘密。好的，那什么是“秘密”？

  | 数据表示
  除必需属性外的设备属性
  支持政策的机制

尽量将未来的变化局限在局部。

隐藏可能独立变更的系统详细信息
在接口中公开不太可能改变的假设条件

- 使用函数来适应变化。

它们比可见的表象更容易改变。

大纲

面向对象风格

目标

- 帕纳斯：隐藏秘密（不只是表象）
- 博奇：对象的行为特征在于它所遭受的动作以及它所要求的动作。

实际上：

对象具有状态和操作，同时还负责其状态的完整性。

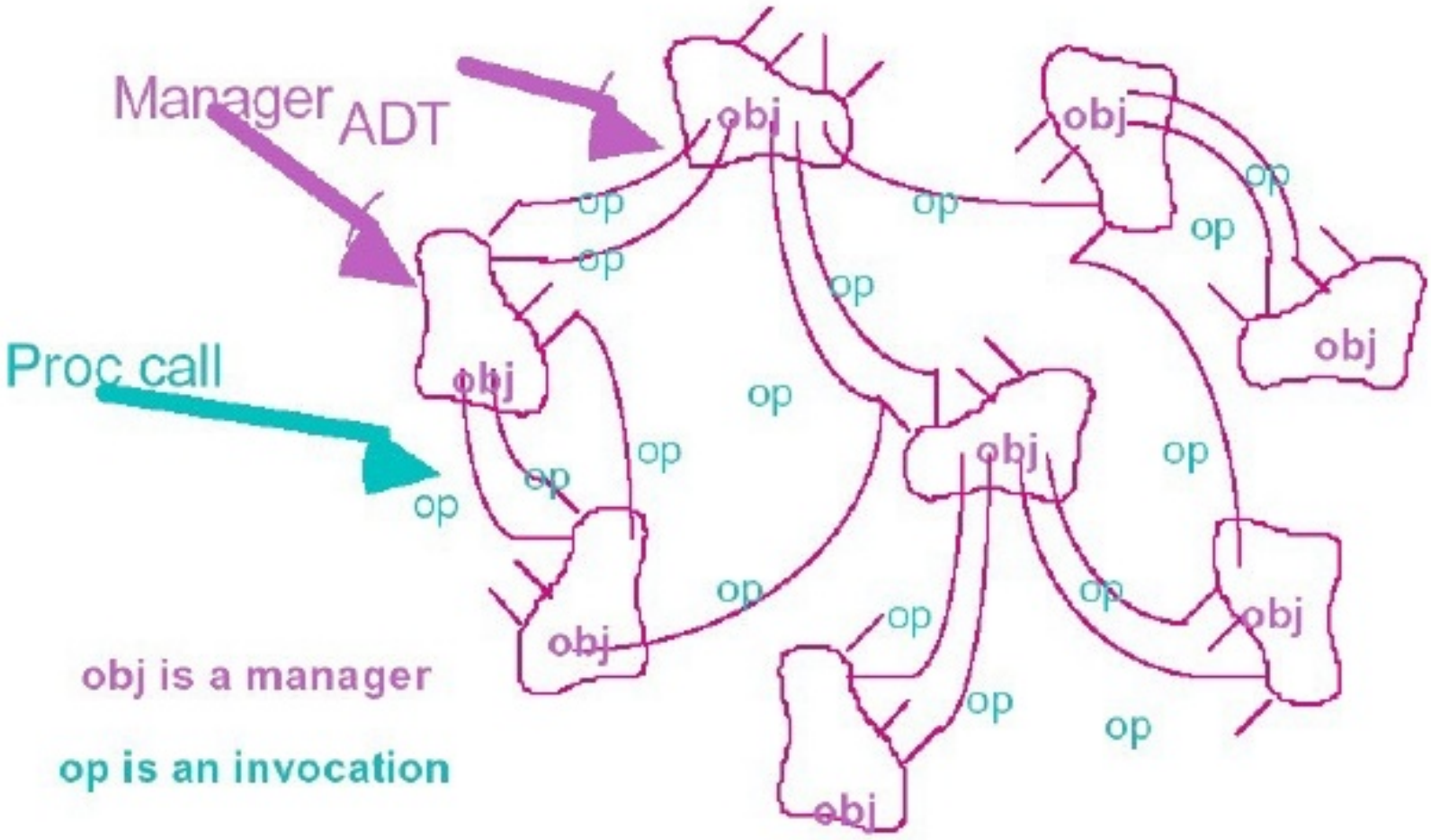
对象通过其接口为人所知。

对象可能是从模板实例化的

对象具有访问和更改状态的操作，也许还有生成器。

存在不同种类的对象（例如，演员、代理、服务器）

面向对象风格


Manager ADT
Proc call
obj
op
obj is a manager
op is an invocation

面向对象风格

问题：这种模式适用于这样的应用场合，即其中的核心问题是识别和保护相关的信息集，尤其是表示信息。

许多设计方法提供了识别自然对象的策略。较新的编程语言支持这一主题的各种变体，因此如果语言选择或方法论已确定，这将对分解的风格产生重大影响。

面向对象风格

解决方案：

- 系统模型：本地化状态维护
- 组件：管理器（例如，服务器、对象、抽象数据类型）
- 连接器：过程调用
- 控制结构：分散式，通常是单线程

面向对象的特性

- **封装：**限制对某些信息的访问

交互：通过过程调用或类似的协议

多态：在运行时选择具体的方法

- **继承：**对共享的功能保持唯一的接口

- **复用与维护：**利用封装和局部性提高生产力

面向对象的问题

管理多个对象

海量的物品需要进一步的分类整理。
- Booch 和 Parnas 提出的分层设计

管理众多互动

单一的界面可能会有局限性且难以操作（因此有了“好友”功能）
- 某些语言/系统允许存在多种接口（内部类、接口、多重继承）

行为的分散责任

使系统难以理解
- 交互图现已用于设计中

面向对象的问题

捕捉相关设计系列

- 类型/类别往往不够用
- 设计模式作为一种新兴的分支

纯粹的面向对象设计会导致出现包含大量对象的大型扁平系统。

同样的老问题可能会再次出现。

- 数百个模块 -> 很难找到东西
- 需要一种施加结构的方法

需要更多的条理和自律

解决方案

客户端-服务器

进程即对象

不对称：客户端了解服务器的情况，而服务器端并不了解客户端。

分层的

对客户端 - 服务器（C/S）模式的详细阐述

- 运行时层的聚合

通常只有少量的层

大纲

分层风格

问题：这种风格适用于涉及可分层排列的不同服务类别的应用程序。通常会有提供基本系统级服务的层、适用于许多应用程序的实用工具层以及特定应用程序任务层。

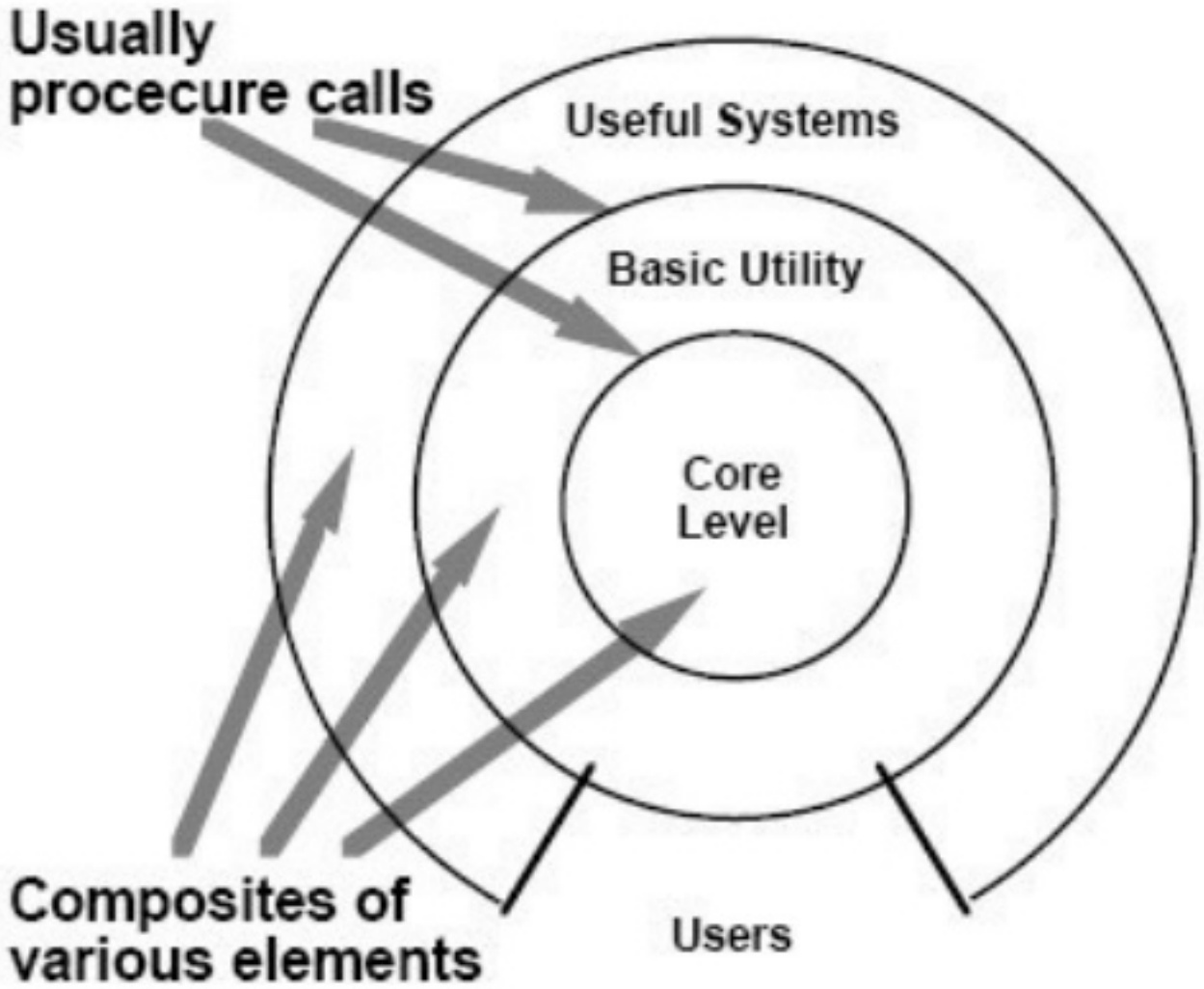
通常，每种服务类别都会分配到一个层级，并且会使用几种不同的模式来细化各个层级。层级最常用于设计的较高层次，通过不同的模式来细化这些层级。

分层风格

解决方案：

- 系统模型：不透明层的层级结构
- 组件：通常是复合体；复合体通常是程序集合
- 连接器：取决于组件的结构；通常在受限的可见性下进行过程调用，也可能是客户端/服务器模式
- 控制结构：单线程

分层风格


Usually
procecure calls
Useful Systems
Basic Utility
Core
Level
Composites of
various elements
Users

分层系统风格的优点

1. 每一层都选择为一组相关的服务；因此，该架构在每一层内都具有很高的内聚性。

2. 每一层都可能对其他层隐藏私有信息。

3. 各层只能使用其下层，从而限制了耦合程度。

易于添加和/或修改当前图层

一层的变化最多影响相邻的两层。

4.每一层都具有内聚性，并且仅与下层耦合，这使得其他人在复用时更加容易，也便于替换或互换。

数据库的变更仅影响数据存储/访问层，浏览器的变更仅影响表示层。

分层系统风格的缺点

1. 严格的分层风格可能会因层级数量的不同而引发性能问题。

如果任何一层仅使用其直接下一层，则为严格分层样式。

如果某一层可以使用其下方的任意一层，那么它就是一种宽松层样式。

2. 要清晰地分层结构化并非总是易事。

3. 各层之间相互调用，影响性能。

大纲

客户端/服务器（C/S）模式

组件是客户端和服务器。

| 客户端：一种应用程序，它向服务器发出请求，并处理与系统环境的输入/输出。

| 服务器：一种响应客户端请求的应用程序。

连接器是基于远程过程调用（RPC）的网络交互协议。

连接器是一种基于 RPC 的网络交互协议。

| RPC（远程过程调用）

客户端-服务器模式

为何选择客户端/服务器模式?

C/S 软件架构基于资源的不均衡性而提出，旨在实现资源共享，于 20 世纪 90 年代趋于成熟。

- 多个用户想要共享和交换数据（网络应用）

- 典型应用领域：分布式多用户（业务）信息系统

C/S 架构的特点

1. 依赖关系：客户端依赖于服务器。

如果任何一层仅使用其直接下一层，则为严格分层样式。

如果某一层可以使用其下方的任意一层，那么它就是一种宽松层样式。

2. 拓扑结构

一个或多个客户端可以连接到一个服务器。

客户之间没有任何联系。

3. 移动性：轻松支持客户端移动性。

4. 安全性：通常在服务器端进行控制，也有可能在应用程序/业务层进行控制。

客户端/服务器架构的优势

1. 能有效利用网络系统。可能需要更便宜的硬件。

2. 易于添加新服务器或升级现有服务器。

3. 允许多个用户之间共享数据。

4. 可扩展：添加新客户。

客户端/服务器架构的缺点

1. 每台服务器中的冗余管理。

2. 很难弄清楚有哪些服务器和可用服务。没有集中登记的名称和服务目录。

3. 服务器和客户端的功能难以更改。

如果应用程序逻辑分布在客户端和服务器端，那么更改它会很困难。

4. 应用程序的可扩展性受服务器和网络容量的限制。

两层客户端/服务器架构

处理管理在用户系统界面环境和数据库管理服务器环境之间进行划分。

- 第 1 层：用户系统界面：在用户的桌面环境中。
- 第 2 级：数据库管理服务：在服务器中。

服务器负责数据管理，客户端完成与用户的交互任务。“胖客户端，瘦服务器”。

两层客户端/服务器架构

限制。

对客户端软件和硬件配置要求高，客户端臃肿

复杂的客户端程序设计

数据安全性很差。客户端程序可以直接访问数据库服务器。

| 单一信息内容与形式

| 用户界面风格多样，使用复杂，无促销用途

软件维护和升级很困难。每个客户端上的软件都需要维护。

三层客户机/服务器架构

与两层的 C/S 结构相比，增加了一个应用服务器。

整个应用程序逻辑都位于应用服务器上，只有表示层存在于客户端：“瘦客户端”。

应用程序功能分为三层：表示层、应用（逻辑）层和数据层。

表示层是应用程序的用户界面部分。通常使用图形用户界面。

应用程序层是应用程序的主要部分，实现具体的业务处理逻辑。

数据层是数据库管理系统。

上述三个层次在逻辑上是相互独立的。

通常在客户端仅配置表示层。

浏览器/服务器架构

B/S 架构是三层 C/S 架构的一种特殊情况。客户端有一个 HTTP 浏览器。

为了增强功能，通常需要安装 Flash、JVM 以及一些特殊的插件。

使用标准的 http/https 协议，省去很多麻烦。

只能“拉”，不能“推”。

客户端之间的通信只能通过服务器进行中转。

浏览器/服务器架构

对客户端资源和其他网络资源的使用有限。

B/S 结构的安全性难以把控（SQL 注入攻击……）。

基于 B/S 结构的应用系统在数据查询及其他相应速度方面远低于 C/S 架构。

服务器负载很重，而客户端的资源却被浪费了。

使用 JVM、Flash、ActiveX 及其他客户端计算技术来解决。

谢谢你!